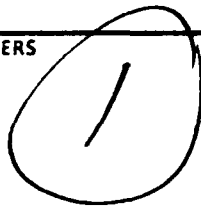
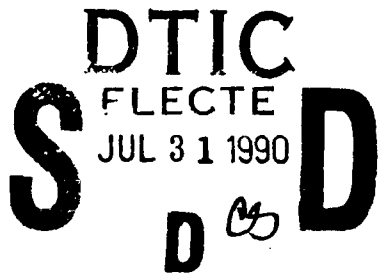


AD-A224 641

DTIC FILE COPY

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 1990	3. REPORT TYPE AND DATES COVERED Thesis/Dissertation	
4. TITLE AND SUBTITLE DATA SECURITY AND INTEGRITY IN OPEN NETWORKS: A PROTOTYPE IMPLEMENTATION OF INTERNET STANDARD PRIVACY-ENHANCED ELECTRONIC MAIL			5. FUNDING NUMBERS 	
6. AUTHOR(S) GORDON D. WISHON				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT Student at: Wright State Univ			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/CI/CIA - 90-045	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFIT/CI Wright-Patterson AFB OH 45433			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release IAW AFR 190-1 Distribution Unlimited ERNEST A. HAYGOOD, 1st Lt, USAF Executive Officer, Civilian Institution Programs			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) 				
14. SUBJECT TERMS			15. NUMBER OF PAGES 332	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	

DATA SECURITY AND INTEGRITY IN OPEN NETWORKS:
A PROTOTYPE IMPLEMENTATION OF INTERNET
STANDARD PRIVACY-ENHANCED
ELECTRONIC MAIL

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

GORDON D. WISHON
B.S.C.S., West Virginia University, 1977

1990
Wright State University

90 07 31 05 6

WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

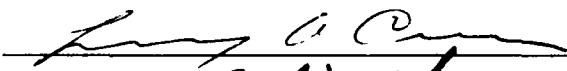


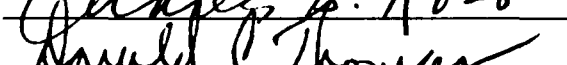
June 21, 1990

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY Gordon D. Wishon ENTITLED
Data Security and Integrity in Open Networks: A Prototype Implementation
of Internet Standard Privacy-Enhanced Electronic Mail
BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF Master of Science.


Thesis Director


Department Chair

Committee on
Final Examination





Dean of the School of Graduate
Studies



Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

Abstract

Wishon, Gordon Duane. M.S., Department of Computer Science and Engineering, Wright State University, 1990. Data Security and Integrity in Open Networks: A Prototype Implementation of Internet Standard Privacy-Enhanced Electronic Mail.

Data security and integrity are crucial issues for virtually every private corporation and government agency using data communications networks to transfer information from one point to another. This is especially true for agencies which employ distributed databases, electronic mail, and electronic funds transfers in their everyday activities. For most agencies, information is an asset to which value can be attributed, and the loss of assets which accompany the loss (through disclosure) of information can indeed be great. In this paper, a prototype implementation of a proposed standard for providing data security and integrity in an electronic mail system will be presented. The paper begins by examining some of the threats to data integrity and security in computers and networks. Next, methods of protecting against such threats are reviewed, and one method, data encryption, will be shown as having widespread popularity over other methods. The Data Encryption Standard (DES) is reviewed, and difficulties with its use are discussed. A recent invention, public key cryptology, is then reviewed in detail, and reasons for its preference over DES are presented. Next, the RSA Public Key Cryptosystem is briefly reviewed, and its use in a proposed standard for providing privacy and data integrity in an electronic mail system is discussed. A prototype implementation of the proposed Internet-standard privacy enhanced electronic mail is then presented. Finally, suggestions for further work are given.

Table of Contents

1	Security in Data Networks	1
1.1	Introduction	1
1.2	Network Security and its Relationship to Computer Security	2
1.3	Additional Characteristics of Networks and Security	5
1.4	Security Services	6
2	Threats to Data Security in Networks	8
2.1	Introduction	8
2.2	Categories of Threats	8
2.3	Protection Mechanisms	10
3	Encryption as a Method of Protecting Data in Networks	13
3.1	Introduction	13
3.2	Link level and end-to-end encryption	14
3.3	Additional Uses of Encryption	16
4	Symmetric Cryptosystems	18
4.1	Introduction	18
4.2	The Data Encryption Standard (DES)	19
4.3	Limitations of Single-key Cryptosystems	20

5	Asymmetric Cryptosystems	21
5.1	Introduction	21
5.2	The RSA Public Key Cryptosystem	23
5.3	Other Public Key Systems	24
5.4	Limitations of the RSA System	25
6	Hybrid Cryptosystems	26
6.1	Key Management	26
6.2	Combining Symmetric and Asymmetric Cryptosystems	28
6.3	Message Digests	28
7	A Proposed Standard for Privacy-Enhanced Electronic Mail	30
7.1	Overview	30
7.2	Description of Approach	31
7.3	Key Management	33
8	A Prototype Implementation of Privacy Enhanced Electronic Mail	35
8.1	Overview	35
8.2	Implementation Environment	36
8.3	Constraints on Development	36
8.4	Limitations of the Implementation	38
8.4.1	Certificate Generation	38
8.4.2	Pemail	39
8.5	General Implementation Design Features	40

8.6	Use of the Privacy Enhanced Electronic Mail System	41
8.6.1	The makekeys Program	41
8.6.2	The cert Program	42
8.6.3	The pemail Program	43
9	Suggestions for Continued Development	46
A	RFC1113 Privacy Enhancement for Internet Electronic Mail: Part I – Message Encipherment and Authentication Procedures	48
B	RFC1114 Privacy Enhancement for Internet Electronic Mail: Part II – Certificate-Based Key Management	129
C	RFC1115 Privacy Enhancement for Internet Electronic Mail: Part III – Algorithms, Modes, and Identifiers	187
D	Source Code and Description – Makekeys.c	206
E	Source Code and Description – Cert.c	219
F	Source Code and Description – Pemail.c	244
G	Source Code and Description – Pesupport.c	300
H	Source Code and Description – Header Files	313
	Bibliography	331

Chapter 1

Security in Data Networks

1.1 Introduction

Abrams and Podell in [1] describe the value of information and the necessity for computer and network security in the following extract. "Computers process and store information. (Computer networks move information from one location to another.) Value may be attributed to this information, which is an asset. The determination of value is both complex and controversial and it may be based on either the cost of developing or assembling the information or on the (potential) loss that would occur if the information were disclosed. The value may be established by the organization that created or gathered the information or it may be established by someone or something else who would like to gain access to the information...". "Much information is considered 'secret' by its owner, whose objective is to maintain its 'confidentiality' against the risk of 'disclosure'. Another important property of information is its 'integrity'; there should be assurances that no inadvertent changes have occurred, causing 'corruption' of the information. The value of some information depends on its source. Its value is enhanced by 'attribution'; it is decreased by the possibility of 'repudiation'. The continuous 'availability' of information and the computer resources to process it are important; 'unauthorized use' and 'denial of service' to authorized users are to be prevented".

In this paper, I will present a prototype implementation of a method for protecting information traveling via electronic mail across a certain class of networks (those employing the Simple Mail Transport Protocol, or SMTP), a method which has been proposed as a standard for the Internet community. This prototype implementation will be presented after a discussion of network security and encryption, and a look at the proposed Internet standard.

1.2 Network Security and its Relationship to Computer Security

Summers in [2] describes the objective of computer, and of course, network security as "... to put the hardware, software, and data out of danger of loss." It includes "... the concepts, techniques, and measures that are used to protect computing systems and the information they maintain against deliberate or accidental threats." Abrams and Podell in [1] state "Network security interfaces with computer security, because many computer systems communicate with end users via networks. The differences between computer security and network security are becoming less apparent as the differences disappear between computer and network systems". However, as we shall see, there are characteristics of networks that do not exist in monolithic computer systems, and the criteria applied to computer systems for evaluating and ensuring security are extended to accommodate them.

The Department Of Defense's Trusted Computer System Evaluation Criteria (TCSEC) (aka the "Orange Book") was published by the National Computer Security Center (NCSC) in December 1985 to "provide a means of evaluating specific security features and assurance requirements available in 'trusted commercially available automatic data processing systems' ", subsequently referred to as AIS. Furthermore, "...

The rating scale of the TCSEC extends from a rating which represents a minimally useful level of trust to one for 'state of the art' features and assurance measures." "The philosophy of protection embodied in the TCSEC requires that the access of subjects (i.e., human users or processes acting on their behalf) to objects (i.e., containers of sensitive information) be mediated in accordance with an explicit and well defined security policy".

A security policy is the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information. For a computer or network system to enforce these policies, they are first translated to a formal security model. Many formal models exist, but the most widely-accepted security model, and the one upon which the TCSEC is based, is the Bell-LaPadula model [3]. The implementation of the formal policy model for a network system is known as the Network Trusted Computing Base, or NTCB. The NTCB is the totality of protection mechanisms within a system – including hardware, firmware, and software – the combination of which is responsible for enforcing the security policy [1].

The Bell-LaPadula model is a formal state transition model of security policy which applies primarily to access control, the major component of any security policy. The model can be used to implement a 'reference monitor' which enforces access control by mediating all accesses to objects by subjects. (The reference monitor is an abstract concept which is implemented in a 'security kernel'). Abrams and Podell describe the model thus: "Control of access is defined by the Bell-LaPadula model by the simple security condition, the *-(star)- property (read as star property), and the discretionary security property. The simple security condition is used as the method to control granting a subject read access to a specific object. This condition allows a subject to have read access to an object only if his security level is equal to or greater than the object (i.e., the subject dominates the security level of the object). Write access to an object is allowed by ... the *-property, only if the security level of the

subject is essentially equal to or less than (is dominated by) the security level of the object. There are trusted subjects (i.e., not constrained by the *-property) and untrusted subjects (that are constrained by the *-property). In addition, the *-property protects against compromise of information by software programs with additional (hidden) functions (i.e., Trojan Horse attacks). Also defined into the model is the discretionary security property, which requires that a specific subject be authorized for a particular mode of access required for the state transition."

In summary, computer systems employing the Bell-LaPadula model in their security kernel may implement mechanisms to provide data secrecy, data integrity, system integrity, labeling of data items, discretionary (need-to-know) protection, mandatory protection, accountability of subjects and the actions they initiate, auditing capability, etc. See [5] for a complete discussion of security requirements for computer systems.

While the Bell-LaPadula model is the most widely accepted, and very successfully applied by the NCSC in the TCSEC (for computer systems), many security models have been formulated. In fact, Rushby[4] describes several reasons why the Bell-LaPadula model should not be applied to network systems, particularly distributed computing environments. To accommodate these arguments, the TCSEC continues, "In the context of network systems, there are a number of additional security services that do not normally arise in individual AIS, and are not appropriate to the detailed feature and assurance evaluation prescribed by the TCSEC. The security services ... include provisions for communications security, denial of service, transmission security, and supportive primitives, such as encryption mechanisms and protocols".

1.3 Additional Characteristics of Networks and Security

As mentioned previously, the NCSC conceded there are characteristics of network systems that are not found in monolithic computer system designs, and as a result, the Trusted Network Interpretation contains a second Part which describes the security 'services', relating to these special characteristics. The following discussion is extracted from Part II of the Trusted Network Interpretation.

Protocols - One of these characteristics are protocols, the set of rules and formats that determine the behavior between entities in a network. Their design and implementation is crucial to the correct, efficient, and effective transfer of information among network systems and subsystems. Many network services are implemented with the help of protocols, and failures and deficiencies in the protocol result in failures and deficiencies in the security services supported by the protocol. One class of design deficiency in protocols are those having some form of denial of service as a result. This class includes deadlocks, livelocks, unspecified receptions, lack-of-liveness, and non-executable interactions. Another class of design concerns are typical of protocols that must work despite various kinds of random interference or communication difficulties, such as noise, message loss, or message reordering. A third class of design deficiency might occur in protocols that are expected to work in the presence of malicious interference, such as active wiretapping. Such protocols should have countermeasures against Message Stream Modification (MSM) attacks.

The Encryption Mechanism - Encryption is a pervasive mechanism for many security services, used in both networks and monolithic computer systems. In networks, encryption is a tool for protecting data from compromise or modification attacks. Through its use, release of message content and traffic analysis can be prevented; message stream modification, some denial of service, and masquerading can be de-

tected. Encryption is probably the most important and widely used mechanism when there is a wiretap threat; sometimes it is even confused with being a service [5]. Much more will be said about encryption later.

1.4 Security Services

The NCSC lists the following as security services, which may or may not be available in specific network offerings: Communications integrity, of which authentication, communications field integrity, and non-repudiation are considered parts; denial of service (continuity of operations); compromise protection, including data confidentiality, traffic confidentiality, and selective routing. A discussion of each of these is presented below (from [5]).

Communications integrity is a collective term for a number of security services. These services are all concerned with the accuracy, faithfulness, non-corruptability, and believability of information transfer between peer entities through the communications network. Authentication deals primarily with ensuring that a data exchange is established with the addressed peer entity (and not with an entity attempting a masquerade or a replay of a previous establishment). The network should assure that the data source is the one claimed. Authentication generally follows identification, i.e., establishing the validity of the claimed identity, providing protection against fraudulent transactions. Communications field integrity refers to the protection of any of the fields involved in communications, e.g., headers fields and user data fields, from unauthorized modification. It can be considered identical to data integrity in monolithic computer systems. The network should ensure information is accurately transmitted from source to destination, regardless of the number of intermittent connecting points. Non-repudiation service provides unforgeable proof of shipment and/or receipt of data. It prevents the sender from disavowing a legitimate

message or the receiver from denying receipt.

Denial of service is the traditional identifier of a service that would probably be better identified as assurance of communications availability. A denial of service condition exists whenever throughput falls below a pre-established threshold, or access to a remote entity is unavailable. It also exists when resources are not available to users on an equitable basis. Denial of service detection is highly dependent on data integrity checking/detection mechanisms.

Compromise protection is also a collective term for a number of security services. These services are all concerned with the secrecy, or non-disclosure of information transfer between peer entities through the communications network. The first, data confidentiality, is mainly compromised through passive wiretapping attacks. Passive attacks consist of observation of information on a link. Release of message content to unauthorized users is the fundamental compromise. Traffic flow confidentiality service protects data against unauthorized disclosure by inference, through analysis of message length, frequency, and protocol components (such as addresses). A selective routing service can be used to avoid specific sub-networks, links, or relays that have shown persistent manipulation attacks, or via only physically secure paths, thereby further protecting data against compromise.

Chapter 2

Threats to Data Security in Networks

2.1 Introduction

Now that we have reviewed the components of 'network security', we can examine the threats against network security, and discuss some of the more common counter-measures that have been taken against these threats. Nearly all threats to computer security are found in network security, and then some, as indicated by the additional 'security services' which the NCSC says should be provided beyond normal computer system protection.

2.2 Categories of Threats

Voydock and Kent in [6] list three distinct categories into which potential security violations can be divided: unauthorized release of information; unauthorized modification of information; and unauthorized denial of resource use. Attacks that cause information release are called 'passive' attacks, while those that cause modification of information or denial of resource use are known as 'active' attacks. In a passive attack, the intruder merely observes protocol data units (PDU's in OSI parlance, or

packets) passing on an association (another OSI term, a connection between peer entities), without interrupting the flow. Such intruder observation is termed 'release of message contents'. Even if the data are unintelligible to him, the intruder can observe the protocol control information (header) and thus learn the location and identities of the communicating entities. Finally, the intruder can examine the lengths of PDU's and their frequency of transmission to learn the nature of the data being exchanged. These latter types are usually referred to as 'traffic analysis.'

The intruder can also mount active attacks, performing a variety of processing on PDU's passing on the association. These PDU's can be selectively modified, deleted, delayed, reordered, duplicated, and inserted into the association at a later point in time, or allowed to pass through unaffected. Bogus PDU's can be synthesized and inserted into the association. While all active attacks involve some combination of these methods, the countermeasures employed against them vary with the form of the attack. For this reason, Voydock and Kent in [6] subdivide active attacks into the the following three categories: message stream modification; denial of message service; and spurious association initiation.

Message stream modification includes attacks on the authenticity, integrity, and ordering of the PDU's passing on the association. Attacks on authenticity can be made by modifying the protocol control information so that they are sent to the wrong destination, or by inserting bogus PDU's into an association. Attacks on integrity can be effected by modifying the data portion of PDU's, whereas attacks on ordering can be effected by deleting PDU's or modifying sequencing information in the protocol control portion of PDU's.

Denial of message service comprises attacks in which the intruder either discards all PDU's passing on an association, or simply delays all PDU's going in one or both directions.

Spurious association initiation comprises attacks in which the intruder either

“plays back” a recording of a previous legitimate association initiation sequence or attempts to establish an association under a false identity. Passive attacks are countered by the ‘Compromise Protection’ network services characterized by the NCSC in the Trusted Network Interpretation [5] and mentioned above. Message stream modification and spurious association initiation can be dealt with by the ‘Communications Integrity’ services mentioned above. Finally, denial of service attacks are addressed by the ‘Assurance of Communications Availability’ services.

2.3 Protection Mechanisms

According to the NCSC, physical protection and encryption are the fundamental techniques for protecting data from compromise [5]. (While physical security is outside the scope of this paper, encryption will be discussed at length in a subsequent section.) Inferences established by traffic analysis can be prevented by masking the frequency, length, and origin-destination patterns of communications between protocol entities. According to Voydock and Kent, appropriate link encryption techniques can mask all such patterns and prevent all traffic analysis attacks. This is not true of end-to-end encryption. In addition, traffic padding can be used to provide various levels of protection against traffic analysis. Finally, as mentioned previously, selective routing techniques can dynamically or by prearrangement choose routes so as to use only physically secure sub-networks, relays, or links. The TNI continues, “End-systems may, on detection of persistent manipulation attacks, wish to instruct the network service provider to establish a connection via a different route.” Also, “... there are national laws and network administration policies governing individual privacy rights, encryption, and trans-border data flow. A user in an end system may wish to specify countries through which certain information should not flow”.

Since networks are typically implemented using a set of protocols one of whose

primary functions is to provide data integrity, a well developed and implemented set of protocols goes far towards assisting in providing 'Communications Integrity' service. When good protocols are combined with encryption, a network can effectively ensure that data has not been subject to excessive random errors, line and node outages, hardware/software failures, and active attack. The contribution of encryption will be shown later. The TNI continues, "Peer entity authentication is an appropriate countermeasure against spurious association initiation (masquerading or playback attacks). Authentication usually follows identification, establishing the validity of the claimed identity providing protection against fraudulent transactions. Verification of identity can be effectively accomplished using passwords, if the password mechanism itself is protected by the network. To tie data to a specific origin, implicit or explicit identification information must be derived and associated with data. Authentication can be provided through an alternate communications channel, or a user-unique cryptographic authentication (e.g., employing a public-key cryptosystem). When cryptographic techniques are used, they may be combined with 'handshaking' protocols and 'liveness' assurances. The liveness assurances can be provided by using synchronized clocks, two and three way handshakes, or by non-repudiation services provided by digital signature mechanisms" (to be discussed in more detail).

"Denial of service detection is highly dependent on data integrity checking and detection mechanisms. Other mechanisms relating to data ordering, modification, loss, or replay (e.g., sequence numbers, frame counts) are also measures of denial of service protection. Mechanisms for addressing denial of service are often protocol based and may involve testing or probing. For example, in order to detect throughput denial of service, a process may exist to measure the transmission rates between peer entities under conditions of input queuing. The measured rate can be compared with a predetermined transmission rate to detect a denial of service condition. Another example is a protocol to detect failure to respond within a certain predetermined time between

peer entities. Finally, a request-response mechanism such as 'are-you- there' message exchanges may be employed when the connection is quiescent. Successful prevention of denial of service attacks would also include network design features such as use of active or passive replacement or other forms of redundancy throughout the network components. Reconfiguration to provide network software maintenance and program downloading to network nodes for software distribution, and to provide initialization and reconfiguration after removing failed or faulty components can isolate and/or confine network failures, accommodate the addition and deletion of network components, and circumvent a detected fault. Distribution and flexibility of network control functions, able to respond promptly to changes in network topology and throughput, enhance survivability and continuity of operations".

Integrity and adequacy of control in a network are the keys in coping with denial of service conditions [5]. Network management and maintenance deal with network health, detecting failures, and overt acts that result in denial of or reduced service. Simple throughput may not necessarily be a good measure of proper performance. Loading above capacity, flooding, replays, and protocol retry due to noise in the channel can reduce service below an acceptable level and/or cause selective outages. For these reasons, the network management and control policy/mechanisms should be developed using appropriate models (queuing theoretic models, hierarchical service models, protocol models, or resource allocation models) which can be analyzed for deadlock, liveness, and other security properties.

Chapter 3

Encryption as a Method of Protecting Data in Networks

3.1 Introduction

As mentioned previously, encryption is a fundamental tool for security in data communications. Voydock and Kent [6] go so far as to say it is “the fundamental technique on which all communications security measures are based.” Although its primary purpose is to prevent compromise of information, it can detect the active attacks of message stream modification, denial of service, and spurious association initiation. It can be effectively used to implement many of the security services we have discussed, such as communications field integrity, peer-entity authentication, non-repudiation, etc.

Encryption is a mapping between ‘plaintext’, or a representation of information which is intelligible to everyone who can read the natural language in which the information is written, and ‘ciphertext’, a representation of the same information designed to conceal the information from unauthorized persons. The transformation between plaintext and ciphertext is encryption. The reverse process is called decryption. The function which accomplishes this mapping may have multiple independent variables as input. The difficulty in deducing the cleartext from the ciphertext without knowing certain of these variables is measured as a ‘work factor.’ The work factor is often

expressed in time units on a specific computer performing cryptanalysis. Without going into detail on the mechanics of various cryptographic systems, suffice it to say that there are such systems for which the work factor gives very good confidence that the deduction cannot be made in a reasonable amount of time.

3.2 Link level and end-to-end encryption

Voydock and Kent describe two basic approaches to communications security provided through encryption: link-oriented measures and end-to-end measures [6]. The former provides security by protecting message traffic independently on each communications link, while the latter provides uniform protection for each message all the way from its source to its destination. These two approaches differ not only in their internal implementation characteristics, but also in the nature of the security they provide.

Link-oriented protection measures provide security for information passing over an individual communication link between two nodes, regardless of the ultimate source and destination of the information. Each link corresponds to a Data Link Layer association in the ISO Reference Model. In many cases, the links will be physically unprotected and thus subject to attack. In a network employing link-oriented measures, encryption is performed independently on each communications link. A different encryption key is often used for each link, so that subversion of one link does not necessarily result in release of information transmitted on other links. Since information is not processed as it passes on a link, both the protocol control information and the data in the PDUs can be enciphered. This masks origin-destination patterns. If a continuous stream of ciphertext bits is maintained between nodes, PDU frequency and length information can be masked as well. In this case, all forms of traffic analysis are completely prevented. Using this technique does not degrade the effective bandwidth of the network because it does not usually require transmission

of any additional data; it does, however, entail continuous keystream generation at each node.

Since information is enciphered only on the links and not within the nodes they connect, the nodes themselves must be secure. Although the origin and destination nodes of the network (i. e., the hosts) are assumed to be physically secure, link encryption requires that all intermediate nodes (packet switches, gateways, etc.) be physically secure as well. Not only must they be physically secure, but their hardware and software components must be certified to isolate the information on each of the associations passing through them. Subverting one of the intermediate nodes exposes all the message traffic passing through that node, despite any physical security precautions still in effect at the source and destination nodes.

Link-oriented measures model a network as a collection of nodes joined by communication links, each of which can be individually protected. End-to-end measures, on the other hand, model a network as a medium for transporting PDUs in a secure fashion from source to destination. In keeping with this perspective, end-to-end, or E3, measures protect PDUs in transit between source and destination nodes in such a way that subversion of any of their communication links does not violate security.

There is some flexibility in defining the points at which E3 security measures are implemented: from host to host, from terminal to service host or process, and from process to process. By extending the domain of E3 measures, one can protect more of the path between communicating protocol entities. However, as their domain is extended, the range of hardware and software that must interface with them increases.

Link-oriented security measures can be implemented so that they are almost completely invisible to network users. E3 measures usually extend beyond the communications subnet and thus require a greater degree of standardization in the protocols employed by those users.

A major advantage of E3 measures is that an individual or host can elect to

employ them without affecting other users and hosts; thus, the cost of employing such measures can be more accurately apportioned. Moreover, these measures can be employed not only in packet-switched networks, but in packet-broadcast networks where link-oriented measures are often not applicable. Finally, E3 measures are more naturally suited to user's perceptions of their security requirements. This stems from the fact that they rely on the security of equipment only at the source and destination of an association, while link-oriented measures require that all nodes in the open-system environment also be secure.

3.3 Additional Uses of Encryption

As we have seen, encryption is a tool which can be used as an effective tool in providing network security services. An authentication service can be provided by encipherment or digital signature mechanisms. In conventional single-key cryptosystems, the encryption of a message with a secret key automatically implies data origin authenticity, because only the holder of the key can produce the encrypted form of a message. Encryption, using a public-key encryption system or other digital signature mechanism also provides non-repudiation services. It should also be obvious that encryption, when used in conjunction with robust communication protocols, can provide excellent data integrity services, as well.

In its most basic form, encryption protects against compromise of information (release of message content). The granularity of key distribution is a trade-off between convenience and protection. Fine granularity would employ a unique key for each sensitivity level for each session; coarse granularity would employ the same key for all session during a time period. Depending on the level of confidentiality required, encryption protection can be extended to all user data on a specific protocol level or datagram (in a connectionless environment), or only to specified fields within the

user data of a PDU.

Traffic flow confidentiality service can be effected through use of link level encryption. This is not true with end-to-end encryption, however. The precision with which such analysis can be done depends on the level in which the encryption is performed. For example, if encryption were performed in the presentation layer, an intruder could determine which presentation, session, and transport entities were involved in a given association. Performing encryption in the transport layer would limit the intruder to observing patterns at the network-address level. That is, he could tell which transport layer entities were exchanging messages, but not which (or how many) higher level entities were doing so.

In summary, proper selection and implementation of encryption mechanisms can be used to counter many of the threats to network security. Combining these mechanisms with others which deal with denial of service, playback attacks, access control, etc. will provide a comprehensive, effective approach to network security.

Chapter 4

Symmetric Cryptosystems (Single-Key Cryptosystems)

4.1 Introduction

Modern encryption functions take at least two inputs. One is the plaintext; the other is a key. The encryption key is a set of information which affects the results of the transformation. Conventional single-key cryptosystems are designed to operate with a single key and encryption function which does not change the length of the text. Let

P = Plaintext

C = Ciphertext

k = Key

E = Encryption function

D = Decryption function

$C = E(k1, P)$

$P = D(k2, C)$

In the single-key system,

$$P = D[k, E(k, P)]$$

$$P = E[k, D(k, P)]$$

That is, the order of encryption and decryption is interchangeable. Further, if $E = D$,

$$E[k, E(k, P)] = P$$

$$D[k, D(k, C)] = C$$

The advantages of selecting this class of functions is that only one function needs to be implemented, only one key is needed, and the order of encryption and decryption becomes moot. The key is normally kept secret. The security of such systems resides entirely in the key k . All other components of the system are assumed to be public knowledge. To maintain security, legitimate users of the system must learn k , while preventing others from learning it.

4.2 The Data Encryption Standard (DES)

The public standard for data encryption was initially developed by IBM and was first standardized by the National Bureau of Standards (NBS) as the Data Encryption Standard (DES) Federal Information Processing Standard Publication (FIPS PUB) 46. It has also since been adapted as an ANSI standard (ANSI Standard X9.9 - 1982, Financial Institution Message Authentication (Wholesale)) under the name Data Encryption Algorithm (DEA). There is nothing secret about the DES algorithm itself. It is a single-key cryptosystem in the public domain and has been designed so that there is no known way, through examination of ciphertext, of determining which transformation (of 2^{56} possible transformations) was used in producing that ciphertext. The security of DES depends on the choice of which possible transformation

(or key selection) is used to encrypt or authenticate a particular message.

4.3 Limitations of Single-key Cryptosystems

As previously mentioned, the security of single-key systems depends on the security of the key. The difficulty of distributing keys has been one of the major limitations on use of conventional cryptographic technology [7]. Key management, the procedures and methods that must be in place to generate, distribute, store, and ultimately destroy keys in such a manner that security is maintained, has become a major issue for users of single-key cryptosystems such as DES. (See ANSI X9.17, Financial Institution Key Management (Wholesale) and [8] for discussions of key management).

A second difficulty which has limited the application of conventional cryptography is its inability to deal with the problem of dispute (non-repudiation) [7]. Conventional (cryptographic) authentication systems can prevent third party forgeries, but cannot settle disputes between the sender and receiver as to what message, if any, was sent. In current commercial practice, the validity of contracts and agreements is guaranteed by handwritten signatures. The essence of a signature is that although only one person can produce it, anybody can recognize it. If there is to be a purely digital replacement for this paper instrument, each user must be able to produce messages whose authenticity can be checked by anyone, say an arbitrator, but which could not have been produced by anyone else, especially the intended recipient. In a conventional (single-key) system the receiver authenticates any message he receives from the sender by deciphering it in a key which the two hold in common. Because this key is held in common, however, the receiver has the ability to produce any cryptogram that could have been produced by the sender and so cannot prove that the sender actually sent a disputed message.

Chapter 5

Asymmetric Cryptosystems (Public Key Cryptosystems)

5.1 Introduction

Public key cryptosystems, a concept invented by Diffie and Hellman, and first described in [9], provide a direct solution to the signature problem, as well as greatly simplifying the problem of key distribution. (Message Authentication Codes (MACs) and Manipulation Detection Codes (MDCs) are commonly employed in DES/DEA implementations for authentication and signature purposes.)

Let

ke = Encryption key

kd = Decryption key

The functions are then

$$C = E(ke, P)$$

$$P = D(kd, C)$$

$$P = D[kd, E(ke, P)]$$

in addition,

$$P = E[ke, D(kd, P)]$$

Public key cryptosystems are designed to work with inverse pairs of keys which have the following properties[10]:

Anything encrypted with one key can be decrypted with the other.

Given one member of the pair, the public key, it is infeasible to discover the other, the private key.

By publicly revealing his public key, the user does not reveal an easy way to compute his private key. This means that in practice only he can decrypt messages encrypted with the public key, or compute the private key efficiently. Two key cryptosystems are called public because of the way they are used. (Note that each user can generate his own pair of keys.) Anyone can send a secret message to the holder of the secret decryption key by employing his public encryption key. Only the holder of the private key can decipher the message.

Alternatively, if a user A wishes to send a signed message to user B, he operates on it with his private key kda to produce the signed message $S = E(kda, P)$. kda was used as A's deciphering key when privacy was desired, but is now used as his enciphering or 'signing' key. When user B receives S he can recover P by operating on S with A's public key kea . B saves S as proof that user A sent him the particular message P . If A later disclaims having sent this message, B can take S to an arbitrator who obtains kea and checks that $D(kea, S) = P$ is a meaningful message with A's name at the end, the proper date and time, etc. Only user A could have generated S because only he knows kda , so A will be held responsible for having sent P .

This technique provides unforgeable, message dependent, digital signatures, but allows any eavesdropper to determine P because only the public information kea is needed to recover P from S . To obtain secrecy of communication as well, A can encrypt S with B's public key and send $T = E(keb, S)$ instead of S . Only B knows kdb , so only he can recover S and thence P . B still saves S as proof that user A sent him P .

5.2 The RSA Public Key Cryptosystem

Rivest, Shamir, and Adelman were the first to develop a complete cryptosystem around Diffie and Hellman's method. Described in [11], their method proceeds as follows:

First, represent the message as an integer between 0 and $n - 1$. (Break a long message into a series of blocks, and represent each block as such an integer.) Use any standard representation. The purpose here is not to encrypt the message but only to get into the numeric form necessary for encryption.

Then, encrypt the message by raising it to the e^{th} power modulo n . That is, the result (the ciphertext C) is the remainder when P^e is divided by n .

To decrypt the ciphertext, raise it to another power d , again modulo n . The encryption and decryption algorithms E and D are thus:

$$C = E(P) = P^e \pmod{n}, \text{ for a message } P.$$

$$D(C) = C^d \pmod{n}, \text{ for a ciphertext } C.$$

Note that the encryption does not increase the size of a message; both the message and the ciphertext are integers in the range 0 to $n - 1$.

The encryption key is thus the pair of positive integers (e, n) . Similarly, the decryption key is the pair of positive integers (d, n) . Each user makes his encryption

key public, and keeps the corresponding decryption key private.

To choose encryption and decryption keys using their method, one first computes n as the product of two primes p and q :

$$n = p * q.$$

These primes are very large, “random” primes. Although n will be made public, the factors p and q will be effectively hidden from everyone else due to the enormous difficulty of factoring n . This also hides the way d can be derived from e .

The integer d is then chosen to be a large, random integer which is relatively prime to $(p - 1) * (q - 1)$. That is, check that d satisfies:

$$\text{gcd}(d, (p - 1) * (q - 1)) = 1 \text{ (“gcd” means “greatest common divisor”).}$$

The integer e is finally computed from p , q , and d to be the “multiplicative inverse” of d , modulo $(p - 1) * (q - 1)$. Thus we have

$$e * d \equiv 1 \pmod{(p - 1) * (q - 1)}.$$

See [11] for the underlying mathematics, algorithms, and correctness proofs. It also discusses methods of finding large primes, and methods for choosing d , and how to compute e from d and θn (the Euler totient function giving the number of positive integers less than n which are relatively prime to n).

5.3 Other Public Key Systems

According to Diffie [10], the strength of RSA has not been proven equivalent to factoring. There might be some method of taking the e^{th} root of M^e without calculating d and thus without providing information sufficient to factor. While at MIT in 1978, Diffie says, M. O. Rabin produced a variant of RSA, subsequently improved by Hugh Williams of the University of Manitoba, that is equivalent to factoring.

Another public key system, due to McEliece, makes use of Goppa codes, for which a fast decoding algorithm is known, according to Diffie. His idea was to construct a Goppa code and disguise it as a general linear code, whose decoding problem is NP-complete. McEliece's system has never achieved widespread acceptance, perhaps due to the size of the keys, which are on the order of a million bits, or due to the substantial expansion of the data entailed in the transformation.

5.4 Limitations of the RSA System

The RSA system makes use of the fact that finding large (e.g. 200 digit) prime numbers is computationally easy, but that factoring the product of two such numbers appears computationally infeasible. At present, however, the convenient features of public key cryptosystems are bought at the expense of speed. Diffie [10] points out that the fastest RSA implementations run at only a few thousand bits per second, while the fastest DES implementations run at many millions. It is generally desirable, therefore, to make use of a hybrid in which the public key systems are used only during key management processes to establish shared keys for employment with conventional systems.

Chapter 6

Hybrid Cryptosystems

6.1 Key Management

As Diffie explains in [11], the solution to the problem of key management using conventional cryptography is for the network to provide a key distribution center (KDC): a trusted network resource that shares a key with each subscriber and uses these in a bootstrap process to provide additional keys to the subscribers as needed. Introducing the concept of a certificate (a cryptographically authenticated message containing a cryptographic key), a user A wishing to communicate with a user B first contacts the KDC and requests a key for communicating with B. The KDC responds by sending A a pair of certificates. Each contains a copy of the required session key, one encrypted so that only A can read it, the other encrypted so that only B can read it. When A calls B, he presents the proper certificates as introduction. Each of them decrypts the appropriate certificate under the key that he shares with the KDC and thereby gets access to the session key. A and B can now communicate securely using the session key.

In order to compromise a network that employs conventional cryptography, it suffices to corrupt the KDC, according to Diffie. This gives the intruder access to information sufficient to recover the session keys used to encrypt past, present, and perhaps future messages. These keys, together with information obtained from passive

wiretaps, allow the penetrators of the KDC access to the contents of any message sent on the system.

A great improvement in both economy and security can be made by the use of public key cryptography. In a conventional network, every subscriber shares a secret key with the KDC and can only authenticate messages explicitly meant for him. The subscribers use this shared secret key to decrypt the session key sent to them by the KDC for use in subsequent communication with other subscribers. However, if one subscriber has the key shared between the KDC and another subscriber, he will be able to intercept and decrypt subsequent messages, as well as forge messages in the name of the other subscriber, or forge messages to the subscriber from the KDC. In a public key system, each subscriber has the public key of the KDC, and thus the capacity to authenticate any message coming from the KDC (using the digital signature), but no power to forge one.

A and B now can create their own key pairs, and request that the KDC include the public components in creating certificates for them. Having each obtained a certificate, they may now communicate by send the other his certificate, authenticate the certificates by checking the KDC's signature on the certificates, and encrypt messages using the key contained in the certificates. When communicating, there is no need to contact the KDC for a session key. The added security arises from the fact that the KDC is not privy to any information that would enable it to spy on the subscribers. The keys that the KDC dispenses are public keys and messages encrypted with these can only be decrypted by using the corresponding private keys, to which the KDC has no access.

Even if the KDC has been corrupted and its private key is known to opponents, this information is insufficient to read the traffic recorded by a passive wiretap. The KDC's private key is useful only for signing certificates containing subscriber's public keys; it does not enable the intruder to decrypt any subscriber traffic. To be able to

gain access to this traffic, the intruders must use their ability to forge certificates as a way of tricking subscribers into encrypting messages with phony public keys.

6.2 Combining Symmetric and Asymmetric Cryptosystems

Taking note of the fact that a disadvantage of public key cryptology is speed, the subscribers may now use their secure communications to exchange their own session keys, which can be implements of a fast encryption method, such as DES. The message can then be encrypted using a secret key, which is then encrypted using the public key contained in the recipient's authenticated certificate, and the encrypted session key appended to the ciphertext. This message can now be transmitted to the recipient, who first decrypts the session key using his private key, and then uses the session key to decrypt the ciphertext.

The above hybridization of conventional and public key cryptosystems is being used in the operation of a secure telephone currently under development at Bell-Northern Research, intended for use on the Integrated Digital Services Network. A similar scheme is also being proposed for use by the Internet community as a proposed standard for privacy enhanced electronic mail, the subject of the rest of this paper.

6.3 Message Digests

Before beginning discussion of privacy enhanced electronic mail, it is worth noting another valuable ingredient of public key cryptology: the message digest. Implementing a digital signature by encrypting the entire document to be signed with a private

key has two disadvantages, as Diffie point out. Because public key systems are slow, both the signature process (encrypting the message with a private key) and the verification process (decrypting the message with a public key) are slow. In addition, if the signature process encrypts the entire message, the recipient must retain the ciphertext for however long the signed message is needed. In order to make any use of it during this period, he must either save a plaintext copy as well or repeatedly decrypt the ciphertext.

Davies and Price proposed constructing a cryptographically compressed form, or digest, of the message and signing by encrypting this value with the private key. In addition to its economies, this has the advantage of allowing the signature to be passed around independently of the message. This is often valuable in protocols in which a portion of the message that is required in the authentication process is not actually transmitted because it is already known to both parties.

Chapter 7

A Proposed Standard for Privacy-Enhanced Electronic Mail

7.1 Overview

In August 1989, the Internet Advisory Board's Privacy Task Force issued a series of Requests for Comments (RFCs) (see Appendices), which suggest draft standard elective protocols for the Internet community in support of privacy enhanced electronic mail. RFC1113, authored by John Linn, describes message encipherment and authentication procedures. RFC1114, authored by Steve Kent and John Linn, describes the certificate based key management to be used, and RFC1115, authored by John Linn, lists the algorithms, modes, and identifiers to be used in mail header fields. In April 1990, a subsequent RFC was released which provided details of paper and electronic formats and procedures for the key management infrastructure being established in support of these services.

This set of proposed standard protocols provides privacy enhancement services for electronic mail transfer within the Internet. These services are offered through the use of end-to-end encryption between originator and recipient User Agent processes, with no special processing requirements imposed on the Message Transfer System at endpoints or intermediate relay sites. The procedures are intended, according to the authors, to be compatible with a wide range of key management approaches, including

both symmetric and asymmetric approaches for encryption of data encryption keys, that is, they support single key cryptosystems and hybrid cryptosystems in the manner discussed in the previous chapter. The privacy enhancement services provided by the protocols are confidentiality (disclosure protection), sender authenticity (authentication), message integrity assurance, and in the hybrid approach, non-repudiation of origin.

7.2 Description of Approach

The following discussion is paraphrased from the RFCs and will include only a description of the hybrid cryptosystem approach, since that is the approach used in the prototype implementation offered in later chapters of this paper.

In this system, two types of keys are employed, as discussed in the previous chapter. A Data Encrypting Key (DEK) is used for encryption of message text. DEKs are generated individually for each transmitted message. Interchange Keys (IKs) are used to encrypt DEKs for transmission within messages. Ordinarily, the same IK will be used for all messages sent from a given originator to a given recipient over a period of time. Each transmitted message contains a representation of the DEK used for message encryption, encrypted under an individual IK per named recipient. The representation is associated with "X- Sender-ID:" and "X-Recipient-ID:" header fields, which allow each individual recipient to identify the IK used to encrypt DEKs for that recipient's use. Given an appropriate IK, a recipient can decrypt the corresponding transmitted DEK representation, yielding the DEK required for message text decryption.

When privacy enhancement processing is to be performed on an outgoing message, a Data Encrypting Key (DEK) is generated for use in message encryption. IK components (public keys) are selected for each individually named recipient; a corre-

sponding "X-Recipient-ID:" field, interpreted in the context of a prior "X-Sender-ID:" field, serves to identify each IK. Each "X-Recipient-ID:" field is followed by an "X-Key-Info:" field, which transfers a DEK, encrypted under the IK appropriate for the specified recipient. A prior "X-MIC-Info:" field carries the message's Message Integrity Check (MIC) quantity (a value obtained by encrypting a message digest with the sender's private component, as discussed in the previous chapter).

A four phase transformation procedure is employed in order to represent encrypted text in a universally transmissible form and to enable messages encrypted on one type of host computer to be decrypted on a different type of host computer. A plaintext message is accepted in local form, using the host's native character set and line representation. The local form is converted to a canonical message text representation, defined as equivalent to the inter-SMTP representation of message text. (Simple Mail Transfer Protocol, or SMTP, is the Internet standard mail transfer protocol.) The canonical representation forms the input to the MIC computation and encryption processes.

For encryption purposes, the canonical representation is padded as required by the encryption algorithm. The padded canonical representation is encrypted. The encrypted text is encoded into a printable form. The printable form is composed of a restricted character set which is chosen to be universally representable across sites, and which will not be disrupted by processing within and between Message Transfer Service entities.

The output of the encoding procedure is combined with a set of header fields carrying cryptographic control information. The result is passed to the electronic mail system to be encapsulated as the text portion of a transmitted message.

When a privacy enhanced message is received, the cryptographic control fields within its text portion provide the information required for the authorized recipient to perform MIC verification and decryption of the received message text. First, the

printable encoding is converted to a bitstring. Encrypted portions of the transmitted message are decrypted. The MIC is verified. The canonical representation is converted to the recipient's local form, which need not be the sender's local form.

Detailed descriptions are given in the RFCs for key generation, algorithms, transformation procedures, and header field formation.

7.3 Key Management

RFC1114 describes a key management architecture that specifically employs the RSA cryptosystem. The authors note that it was selected because it provides all the necessary algorithmic facilities, is "time tested" and is relatively easy to implement in either hardware or software. It is also the primary algorithm identified (at this time) for use in international standards where an asymmetric encryption algorithm is required. The architecture is based on the use of certificates, in which a "certification authority", or CA, representing an organization applies a digital signature to a collection of data consisting of a user's public key, various information serving to identify that individual, and the identity of the organization whose signature is affixed. This establishes a binding between the user credentials, the user's public component, and the organization which vouches for this binding. The resulting signed data item is called a certificate. The organization identified as the certifying authority for the certificate is the "issuer" of the certificate.

In signing the certificate, the CA vouches for the user's identification, especially as it relates to the user's affiliation with the organization. The digital signature is affixed on behalf of that organization and is in a form which can be recognized by all members of the privacy enhanced electronic mail community. Once generated, certificates can be stored in directory servers, transmitted via unsecure message exchanges, or distributed via any other means that make certificates easily accessible to message

originators, without regard for the security of the transmission medium.

Prior to sending an encrypted message, an originator must acquire a certificate for each recipient and must validate these certificates. Briefly, validation is performed by checking the digital signature in the certificate, using the public component of the issuer, whose private component was used to sign the certificate.

Once a certificate for a recipient is validated, the public component contained in the certificate is extracted and used as described in the previous section to effect privacy enhanced electronic mail transfer.

In order to provide message integrity and data origin authentication, the originator generates a MIC, signs (encrypts) the MIC using his private component, and includes the resulting value in the "X-MIC-Info:" field as described in the previous section. The certificate of the sender is also included in the header in the "X-Certificate:" field, in order to facilitate validation in the absence of ubiquitous directory services. Upon receipt of a privacy enhanced message, a recipient validates the sender's certificate, extracts the public component from the certificate, and uses that value to recover (decrypt) the MIC. The recovered MIC is compared against the locally calculated MIC to verify the integrity and data origin authenticity of the message.

RFC1114 describes in detail the generation of certificates, and suggests a hierarchy of certification organizations in which the complexity of validation is minimized by limiting the length of certification "paths". In addition, the authors suggest methods for interoperation across certification hierarchy boundaries (see Appendices for details).

Chapter 8

A Prototype Implementation of Privacy Enhanced Electronic Mail

8.1 Overview

RSA Data Security, Inc., founded by Rivest, Shamir, and Adelman, have produced a set of embeddable routines called BSAFE which provide primitives of the RSA public key cryptosystem. This package was made available under license to the U.S. Government, and was provided to me through the Office of the Secretary of Defense, Automation Support & Technology, for research purposes. Using RFC1113 and RFC1114 as design specifications, I have developed a prototype implementation of the proposed standard privacy enhanced electronic mail, built around the set of BSAFE primitives. Before discussing details of the implementation, the implementation environment, constraints on development, and limitations of the implementation will be presented. Also, please note that the BSAFE routines are protected by copyright, and as such, will not be reproduced in this work.

8.2 Implementation Environment

The electronic mail implementation was developed using the C programming language on a Vax 11/785 running Mt Xinu's 4.3bsd Unix operating system. Certificate generation is most secure when performed in a stand alone environment, and so the certificate generation process was developed on a Zenith Z-184 running MS-DOS 3.2 under the Microsoft C programming language, version 5.0. These machines use the same byte-ordering, and all code in this implementation is portable between these types of machines. Furthermore, compile time options were used which allow for compiling on M68000 based machines (Suns, etc.), which use a different byte-ordering, and all code should be portable to these machines as well, although this porting was not accomplished during the project.

BSAFE Version 1.4, a software package consisting of a set of subroutines which provide a variety of cryptographic functions, was provided by RSA Data Security, Inc. Note: Calls to the BSAFE routines will be shown in the subsequent description of the implementation, but due to copyright restrictions, the subroutines themselves will not be duplicated in this paper. The BSAFE routines were compiled and built into an object library under both the Unix and MS-DOS operating systems.

8.3 Constraints on Development

It is the intent of the authors of the proposed standards that any real world implementation of privacy enhanced electronic mail be implemented using ASN.1 syntax and encoding mechanisms, in which all defined data fields are self-describing, self-delimiting and inherently of variable length. In using ASN.1 encoding, lengths of data items transmitted from one machine to another are known to the receiving machine through the encoding mechanism, and need not be of predetermined size. An

ASN.1 compiler was simply not available during the implementation period, and was a major constraint, forcing the addition of certain non standard data fields to those defined in the RFCs to accommodate lengths of variable length fields, such as message texts.

In addition, while the proposed standards allow for the use of modulus lengths of between 320 and 632 bits, I chose the smallest size possible, 320 bits, in order to speed up the generation of keys, and to allow for a predictable, fixed length field to be used for key transmission (in the generation of certificates, for example).

The implementation followed the suggestion of the developers of the proposed standards in that the privacy enhanced mail program is a standalone program which is invoked by a user, and lies above the existing User Agent sublayer. This form of integration offers, as the authors point out, the advantage that the program can be used in conjunction with a wide range of User Agent programs, rather than being compatible with only a particular User Agent.

When a user wishes to apply privacy enhancements to an outgoing message, the user prepares the message's text and invokes the standalone program (interacting with the program to provide address information and other data to perform privacy enhancement processing), which in turn generates output suitable for transmission via the User Agent. When a user receives a privacy enhanced message, the User Agent delivers the message in encrypted form, suitable for decryption and associated processing by the standalone program.

The final constraint was time, and due to never having as much as one would like, forced me to restrict the implementation to something I felt was "do-able" in the time I had available. Therefore, the implementation of certain features, such as certificate validation, will be left to those who follow. These, and other limitations of the implementation, are discussed in the next section.

8.4 Limitations of the Implementation

Due to time constraints, a number of limitations exist in the current prototype implementation. The following sections describe these limitations. A fully functioning implementation would have to address each of these limitations in turn:

8.4.1 Certificate Generation

A method is required to deal with the problem of compromised and otherwise invalid certificates. Some means of notifying users of the privacy enhanced electronic mail system about invalid certificates must be provided, as well as a method of maintaining a list of invalid certificates which can be checked upon receipt of a privacy enhanced message. RFC1114 describes the use of a Certificate Revocation List for this purpose. Certificate revocation has not been addressed in my implementation.

Serial numbers for certificates are required to be unique, and RFC1114 suggests the use of monotonically increasing integers for this purpose. This implementation requires the generator of the certificate to enter a serial number in response to a prompted message, and no provisions are made to insure uniqueness.

The certificate generation routines require calls to a random number generator, which gets its seed by a call to the system clock. Since certificate generation is envisioned to be performed only on a standalone device such as a PC, only MSDOS system calls were used. If another platform is used, appropriate changes to the code are required.

The prototype implementation prompts the user for the name of the file in which the appropriate certificate is stored. A more satisfying implementation would place all certificates in a central cache on each host (or in a server).

8.4.2 Pemail

A major component of security of the privacy enhanced mail system lies in its ability to ensure the authenticity of the receiver, and to ensure the non-repudiation of origin of the message. Authenticity of the recipient is accomplished by the sender validating the certificate of the recipient prior to sending him mail. Non-repudiation of origin is accomplished by ensuring the digital signature of the message matches the one produced by the recipient. Part of this assurance process is the validation of the sender's certificate. The process of validating certificates involves little more than ensuring a digital signature produced locally matches the one on the certificate, decrypted using the public component of the Issuing Authority (see RFC 1114 for details). Although provisions are made in my implementation for certificate validation, at the time of this writing it is incomplete.

This implementation provides no support for multiple recipients and mailing lists. Multiple recipients can be accommodated by repeating the encryption of the DEK using the public component of each recipient, and including an "X-Recipient-ID" and "X-Key-Info" header for each recipient. Note that the message text only need be encrypted once (assuming there is no part of the message to be kept private from a member of the recipient list). RFC1113 suggests two methods of providing support for mailing lists.

The sender must determine if recipient is capable of processing privacy enhanced email. This may not be assumed even if a certificate for the recipient exists, and is available to the sender. (The recipient may be receiving mail on host which has no privacy enhanced electronic mail processing capability.) In the existing implementation, this is accommodated in a straightforward manner: if the recipient receives a privacy enhanced mail message for which he has no capability to process, he simply notifies the sender of that fact. The sender may choose another method of transmitting the message.

Exclusion of portions of message text from encryption is not supported. This can be accommodated in future implementations through modifications to the code which parses the text message.

8.5 General Implementation Design Features

The prototype implementation of privacy enhanced electronic mail consists of several standalone programs developed to generate sets of keys, to generate user certificates, and to process text messages for sending/receiving by the existing mail User Agent. As a prototype system, all programs were developed in a very simplistic manner, using no coding "tricks" and performing minimal error checking, all with the intent to aid in the understanding of the mechanisms involved. All programs prompt for user input, and accept no command line arguments. BSAFE routines were called in precisely the manner suggested by the BSAFE documentation, and no changes were made to any BSAFE routine. (Header files were, of course, modified as required for the appropriate host architecture and compiler.) The following standalone programs were developed:

1. makekeys – generates public/private key pairs for users, saving them in files of the user's choice.
2. cert – generates user certificates, saving them in files of the user's choice. The reader should review RFC1114 (Appendix B) for details of the certification procedures.
3. pemail – the privacy enhanced mail processing routine. Prepares text messages for sending, saving the processed message in a file to be passed to the User Agent. Also performs processing of incoming privacy enhanced mail, storing the resulting text in a file of the user's choice. The reader should review RFC1113 (Appendix A)

for details of the processing required to transform messages.

In addition, the following support routines were developed to be used by one or more of the above programs, and can be found in the file pesupport.c:

1. pencode – encodes text to printable form in accordance with RFC1113.
2. pdecode – decodes from printable form.
3. printstatus – prints the status returned by any of the other routines.
4. initrandom – initializes a psuedo-random number generator.

8.6 Use of the Privacy Enhanced Electronic Mail System

8.6.1 The makekeys Program

In order to make use of privacy enhanced electronic mail, the user must have the ability to generate public/private key pairs. A user may have several pairs of keys, for various purposes, but the public component of the pair used for privacy enhanced mail processing will be used in the generation of the user's certificate. It is imperative that the corresponding private component be protected by the user, as its compromise will place at risk all subsequent privacy enhanced mail messages, as well as any messages which the user has decided to retain in privacy enhanced form, with encrypted DEK intact. For the purposes of this implementation, we will assume that the file protection capabilities of the host operating system can provide that protection. The makekeys program is simply executed from the command line, with no arguments. The user will be prompted for file names in which to store his public and provate components.

Source Code and Description for the Makekeys program can be found in Appendix D.

8.6.2 The cert Program

Once a user has generated a key pair, the public component is used in generating a certificate for that user. A certificate essentially includes information identifying the user, along with his public component, and a message digest, which serves to bind the identifying information to the public component. This message digest is signed (i.e., encrypted) with private component of the Certifying Authority. The certificate signature can then be used by the user to determine that the integrity of its contents have not been compromised subsequent to generation by the Certifying Authority.

The data fields used to create user certificates are addressed in detail in RFC1114, but procedures for ensuring the security and integrity of the certificate generation process are not. It is important in this system that the certificate generation process be secure and that the integrity of the data (the user's identifying information and his public component) relayed to the issuer of the certificate and back to the user be assured. Forged certificates give an intruder the opportunity to trick a user into encrypting messages with phony public keys.

Note that to forge a certificate, an intruder must discover the private component of the Certifying Authority. This knowledge cannot allow the intruder to directly decrypt any user's traffic; however, if an intruder replaced user A's certificate with a forged certificate whose corresponding private key is known only to the intruder, this would allow the intruder to decrypt any message that user B sends to A.

For the purposes of the prototype, the certificate generation program is executed by the Certifying Authority from a standalone, trusted PC, in order to help protect the Certifying Authority's private component from compromise. Note also that the

Certifying Authority issues certificates on behalf of an organization, using the organization's private component, not his personal key. Also, in this implementation the user's public component and identifying information are simply mailed (not privacy enhanced) to the Certifying Authority, and the resulting certificate is simply mailed back. These procedures would not normally suffice, and procedures described in RFC1114 should be used.

To use the program, the Certifying Authority simply executes the program cert from the command line, with no arguments, and responds to the prompts, which asking him to enter the user's identifying information, and the name of the file in which the user's public component is stored. The identifying information is read into a data structure of fixed size. Certificate specific information (serial number, version number, validity period, etc.) are generated and copied into the structure. The file in which the user's public key is stored is opened, and the key value is recovered. This value is also read into the structure. Next, a message digest, or Message Integrity Code (MIC), is calculated on the contents of the structure, encrypted using the private component of the Certifying Authority, and appended to the structure. The structure contents are then written out to a file. This file is the user's certificate.

Source Code and Descriptions of the Cert program can be found in Appendix E.

8.6.3 The pemail Program

pemail is the main component of the privacy enhanced electronic mail implementation, and like the previous programs, is invoked with no command line arguments, prompting for needed information. The program is broken into two major sections, consisting of modules needed to prepare a message for sending, and modules needed to receive privacy enhanced mail.

The Sending Routines

Preparing a privacy enhanced mail message for sending entails performing the four phase transformation on the message text in accordance with RFC1113. Interchange Keys must be recovered from appropriate certificates and Data Encryption Keys must be generated. After prompting for the file name of the text message and opening the file, a routine is called to transform the message text into a universal canonical form. This conversion to a standard character set and representation before encryption allows a message and its MIC to be verified at any type of destination host computer. The decryption and MIC verification is performed before any conversions which may be necessary to transform a message into a destination-specific local form. (See RFC1113 for details of this transformation.) The canonical form of the message is passed to a routine which calculates a MIC, which will be encrypted using the private component of the sender and transmitted in a header field. This will allow the recipient to verify the received message's integrity. Next, the DEK is generated, and the canonical form of the message text is encrypted using the DEK. After this, the resulting encrypted bitstring is encoded into characters which are universally representable at all sites, by the pencode routine. (See RFC1113 for details on the encoding scheme used, and Appendix G for code and module descriptions of the Pencode and Pdecode routines.) Next, after prompting for the appropriate certificate file names, the header fields are created, recovering and inserting certificates where required. It is in this step that the DEK is encrypted using the recipient's public component (recovered from his certificate) and placed in the "X-Key-Info:" header field. Finally, the header fields and encrypted message text are written to a text file for encapsulation as an electronic mail text message.

The Receiving Routines

The receiving routines essentially perform an inverse transformation, in reverse order, on the text message. Having saved the privacy enhanced mail message in a file through the mail User Agent, the receiving routines first decode the encrypted message text into the local character set. (The recipient may optionally wish to validate the certificate of the sender, contained in the "X-Certificate:" header field.) After decrypting the DEK with the recipient's private component, the message text is decrypted (into its canonical form) and a MIC calculation is performed on the resulting text. This MIC quantity is compared to the decrypted (using the sender's public component recovered from the "X-Certificate:" header field) quantity in the "X-MIC-Info:" header field to verify the integrity of the message transmission. Next, the decrypted text is converted from canonical form to local form and saved in a file.

The specific code and module descriptions of the Pemail program can be found in Appendix F.

Chapter 9

Suggestions for Continued Development

This prototype has shown the feasibility of implementing privacy enhanced electronic mail in a typical open environment. While it may be some time before the proposed standards can be implemented across the Internet, a single organization, issuing its own certificates, could implement such a system in a short time, resulting in significantly improved security for sensitive information. Such an organization would not face many of the constraints one would see in a large entity such as the Internet. (For example, the problem of ensuring unique user names across the entire domain is not insignificant for the Internet community.) Cacheing of certificates on all the organization's hosts could substitute for certificate servers, and a single entity within the organization, say the Security Manager, could be responsible for issuing certificates.

While this prototype could serve as the basis for such an implementation, many areas would need further work and improvement. The primary improvement would be to rework the implementation in ASN.1, to more closely comply with the standards and eliminate the need for fixed length data fields, which required a great deal of extra effort to implement. Error checking would need to be improved, as the prototype is not very robust, and it would be very easy to break by exceeding the bounds of buffers in the C gets function calls. A major limitation of the current prototype is that certificate validation has not been implemented, and this would be a must in a

real-world environment. In general, all the items listed as limitations of the prototype should be addressed.

There are other interesting applications of hybrid cryptosystems to be explored. For example, because of the slow speed of the public key algorithms, one would not wish to use it to protect data on a computer system. However, it would be an easy task to build a package that would encrypt a file using a DES (fast algorithm) key, encrypt the DES key using a public key, and append the encrypted DES key to the file for storage. In this way, a user could protect data stored on, for example, a floppy disk from compromise during transit.

Another application might be in virus protection, using the digital signature capability of public key cryptology. In this application, a software vendor would compute a message digest of his code, encrypt the message digest using his private component, and append the encrypted digest to his product. A customer, purchasing the product, could calculate the message digest, and compare his calculated message digest with the one sent with the product, decrypted using the vendor's published public component. If they were not equal, the customer should suspect that the product had been tampered with. Note, however, that this scheme requires a strong certificate validation mechanism.

In summary, the standards proposed in RFCs 1113-1115, employing a hybrid cryptosystem to provide data confidentiality and integrity across unsecure communications channels, appear to offer a highly effective means to ensure the protection and authentication of electronic mail messages. If certificate management issues can be dealt with effectively, organizations may soon be able to communicate sensitive information without the necessity of costly trusted networks and components.

Appendix A

RFC1113 Privacy Enhancement for Internet Electronic Mail: Part I – Message Encipherment and Authentication Procedures

Network Working Group

J. Linn

Request for Comments: 1113

DEC

Obsoletes RFCs: 989, 1040

IAB Privacy Task Force

August 1989

Privacy Enhancement for Internet Electronic Mail:

Part I -- Message Encipherment and Authentication Procedures

STATUS OF THIS MEMO

This RFC suggests a draft standard elective protocol for the Internet

community, and requests discussion and suggestions for improvements.

Distribution of this memo is unlimited.

ACKNOWLEDGMENT

This RFC is the outgrowth of a series of IAB Privacy Task Force meetings and of internal working papers distributed for those meetings. I would like to thank the following Privacy Task Force members and meeting guests for their comments and contributions at the meetings which led to the preparation of this RFC: David Balenson, Curt Barker, Jim Bidzos, Matt Bishop, Danny Cohen, Tom Daniel, Charles Fox, Morrie Gasser, Russ Housley, Steve Kent (chairman), John Laws, Steve Lipner, Dan Nessel, Mike Padlipsky, Rob Shirey, Miles Smid, Steve Walker, and Steve Wilbur.

Table of Contents

1. Executive Summary	2
2. Terminology	3
3. Services, Constraints, and Implications	3
4. Processing of Messages	7
4.1 Message Processing Overview	7
4.1.1 Types of Keys	7

4.1.2 Processing Procedures	8
4.2 Encryption Algorithms and Modes	9
4.3 Privacy Enhancement Message Transformations	10
4.3.1 Constraints	10
4.3.2 Approach	11
4.3.2.1 Step 1: Local Form	12
4.3.2.2 Step 2: Canonical Form	12
4.3.2.3 Step 3: Authentication and Encipherment	12
4.3.2.4 Step 4: Printable Encoding	13
4.3.2.5 Summary of Transformations	15
4.4 Encapsulation Mechanism	15
4.5 Mail for Mailing Lists	17
4.6 Summary of Encapsulated Header Fields	18

Linn

[Page 1]

RFC 1113

Mail Privacy: Procedures

August 1989

4.6.1 Per-Message Encapsulated Header Fields	20
4.6.1.1 X-Proc-Type Field	20
4.6.1.2 X-DEK-Info Field	21

4.6.2 Encapsulated Header Fields Normally Per-Message	21
4.6.2.1 X-Sender-ID Field	22
4.6.2.2 X-Certificate Field	22
4.6.2.3 X-MIC-Info Field	23
4.6.3 Encapsulated Header Fields with Variable Occurrences	23
4.6.3.1 X-Issuer-Certificate Field	23
4.6.4 Per-Recipient Encapsulated Header Fields	24
4.6.4.1 X-Recipient-ID Field	24
4.6.4.2 X-Key-Info Field	24
4.6.4.2.1 Symmetric Key Management	24
4.6.4.2.2 Asymmetric Key Management	25
5. Key Management	26
5.1 Data Encrypting Keys (DEKs)	26
5.2 Interchange Keys (IKs)	26
5.2.1 Subfield Definitions	28
5.2.1.1 Entity Identifier Subfield	28
5.2.1.2 Issuing Authority Subfield	29
5.2.1.3 Version/Expiration Subfield	29
5.2.2 IK Cryptoperiod Issues	29
6. User Naming	29
6.1 Current Approach	29
6.2 Issues for Consideration	30
7. Example User Interface and Implementation	30

8. Areas For Further Study	31
9. References	32
NOTES	32

1. Executive Summary

This RFC defines message encipherment and authentication procedures, in order to provide privacy enhancement services for electronic mail transfer in the Internet. It is one member of a related set of four RFCs. The procedures defined in the current RFC are intended to be compatible with a wide range of key management approaches, including both symmetric (secret-key) and asymmetric (public-key) approaches for encryption of data encrypting keys. Use of symmetric cryptography for message text encryption and/or integrity check computation is anticipated. RFC-1114 specifies supporting key management mechanisms based on the use of public-key certificates. RFC-1115 specifies algorithm and related information relevant to the current RFC and to RFC-1114. A subsequent RFC will provide details of paper and electronic formats and procedures for the key management infrastructure being established in support of these services.

Privacy enhancement services (confidentiality, authentication, and

Linn

[Page 2]

RFC 1113

Mail Privacy: Procedures

August 1989

message integrity assurance) are offered through the use of end-to-end cryptography between originator and recipient User Agent processes, with no special processing requirements imposed on the Message Transfer System at endpoints or at intermediate relay sites. This approach allows privacy enhancement facilities to be incorporated on a site-by-site or user-by-user basis without impact on other Internet entities. Interoperability among heterogeneous components and mail transport facilities is supported.

2. Terminology

For descriptive purposes, this RFC uses some terms defined in the OSI X.400 Message Handling System Model per the 1984 CCITT Recommendations. This section replicates a portion of X.400's Section 2.2.1, "Description of the MHS Model: Overview" in order to make the terminology clear to readers who may not be familiar with the OSI MHS Model.

In the MHS model, a user is a person or a computer application. A user is referred to as either an originator (when sending a message) or a recipient (when receiving one). MH Service elements define the set of message types and the capabilities that enable an originator to transfer messages of those types to one or more recipients.

An originator prepares messages with the assistance of his or her User Agent (UA). A UA is an application process that interacts with the Message Transfer System (MTS) to submit messages. The MTS delivers to one or more recipient UAs the messages submitted to it. Functions performed solely by the UA and not standardized as part of the MH Service elements are called local UA functions.

The MTS is composed of a number of Message Transfer Agents (MTAs). Operating together, the MTAs relay messages and deliver them to the intended recipient UAs, which then make the messages available to the intended recipients.

The collection of UAs and MTAs is called the Message Handling System (MHS). The MHS and all of its users are collectively referred to as the Message Handling Environment.

3. Services, Constraints, and Implications

This RFC defines mechanisms to enhance privacy for electronic mail transferred in the Internet. The facilities discussed in this RFC provide privacy enhancement services on an end-to-end basis between sender and recipient UAs. No privacy enhancements are offered for message fields which are added or transformed by intermediate relay points.

Linn

[Page 3]

RFC 1113

Mail Privacy: Procedures

August 1989

Authentication and integrity facilities are always applied to the entirety of a message's text. No facility for confidentiality without authentication is provided. Encryption facilities may be applied selectively to portions of a message's contents; this allows less sensitive portions of messages (e.g., descriptive fields) to be processed by a recipient's delegate in the absence of the recipient's personal cryptographic keys. In the limiting case, where the entirety of message text is excluded from encryption, this feature

can be used to yield the effective combination of authentication and integrity services without confidentiality.

In keeping with the Internet's heterogeneous constituencies and usage modes, the measures defined here are applicable to a broad range of Internet hosts and usage paradigms. In particular, it is worth noting the following attributes:

1. The mechanisms defined in this RFC are not restricted to a particular host or operating system, but rather allow interoperability among a broad range of systems. All privacy enhancements are implemented at the application layer, and are not dependent on any privacy features at lower protocol layers.
2. The defined mechanisms are compatible with non-enhanced Internet components. Privacy enhancements are implemented in an end-to-end fashion which does not impact mail processing by intermediate relay hosts which do not incorporate privacy enhancement facilities. It is necessary, however, for a message's sender to be cognizant of whether a message's intended recipient implements privacy enhancements, in order that encoding and possible

encipherment will not be performed on a message whose destination is not equipped to perform corresponding inverse transformations.

3. The defined mechanisms are compatible with a range of mail transport facilities (MTAs). Within the Internet, electronic mail transport is effected by a variety of SMTP implementations. Certain sites, accessible via SMTP, forward mail into other mail processing environments (e.g., USENET, CSNET, BITNET). The privacy enhancements must be able to operate across the SMTP realm; it is desirable that they also be compatible with protection of electronic mail sent between the SMTP environment and other connected environments.
4. The defined mechanisms are compatible with a broad range of electronic mail user agents (UAs). A large variety of

electronic mail user agent programs, with a corresponding broad range of user interface paradigms, is used in the Internet. In order that electronic mail privacy enhancements be available to the broadest possible user community, selected mechanisms should be usable with the widest possible variety of existing UA programs. For purposes of pilot implementation, it is desirable that privacy enhancement processing be incorporable into a separate program, applicable to a range of UAs, rather than requiring internal modifications to each UA with which privacy-enhanced services are to be provided.

5. The defined mechanisms allow electronic mail privacy enhancement processing to be performed on personal computers (PCs) separate from the systems on which UA functions are implemented. Given the expanding use of PCs and the limited degree of trust which can be placed in UA implementations on many multi-user systems, this attribute can allow many users to process privacy-enhanced mail with a higher assurance level than a strictly UA-based approach would allow.
6. The defined mechanisms support privacy protection of

electronic mail addressed to mailing lists (distribution lists, in ISO parlance).

7. The mechanisms defined within this RFC are compatible with a variety of supporting key management approaches, including (but not limited to) manual pre-distribution, centralized key distribution based on symmetric cryptography, and the use of public-key certificates. Different key management mechanisms may be used for different recipients of a multicast message. While support for a particular key management mechanism is not a minimum essential requirement for compatibility with this RFC, adoption of the public-key certificate approach defined in companion RFC-1114 is strongly recommended.

In order to achieve applicability to the broadest possible range of Internet hosts and mail systems, and to facilitate pilot implementation and testing without the need for prior modifications throughout the Internet, three basic restrictions are imposed on the set of measures to be considered in this RFC:

1. Measures will be restricted to implementation at endpoints and will be amenable to integration at the user agent (UA)

level or above, rather than necessitating integration into the message transport system (e.g., SMTP servers).

Linn

[Page 5]

RFC 1113

Mail Privacy: Procedures

August 1989

2. The set of supported measures enhances rather than restricts user capabilities. Trusted implementations, incorporating integrity features protecting software from subversion by local users, cannot be assumed in general. In the absence of such features, it appears more feasible to provide facilities which enhance user services (e.g., by protecting and authenticating inter-user traffic) than to enforce restrictions (e.g., inter-user access control) on user actions.
3. The set of supported measures focuses on a set of functional capabilities selected to provide significant and tangible benefits to a broad user community. By concentrating on the

most critical set of services, we aim to maximize the added privacy value that can be provided with a modest level of implementation effort.

As a result of these restrictions, the following facilities can be provided:

1. disclosure protection,
2. sender authenticity,
3. message integrity measures, and
4. (if asymmetric key management is used) non-repudiation of origin,

but the following privacy-relevant concerns are not addressed:

1. access control,
2. traffic flow confidentiality,
3. address list accuracy,

4. routing control,
5. issues relating to the casual serial reuse of PCs by multiple users,
6. assurance of message receipt and non-deniability of receipt,
7. automatic association of acknowledgments with the messages to which they refer, and
8. message duplicate detection, replay prevention, or other

Linn

[Page 6]

RFC 1113

Mail Privacy: Procedures

August 1989

stream-oriented services.

A message's sender will determine whether privacy enhancements are to be performed on a particular message. Therefore, a sender must be

able to determine whether particular recipients are equipped to process privacy-enhanced mail. In a general architecture, these mechanisms will be based on server queries; thus, the query function could be integrated into a UA to avoid imposing burdens or inconvenience on electronic mail users.

4. Processing of Messages

4.1 Message Processing Overview

This subsection provides a high-level overview of the components and processing steps involved in electronic mail privacy enhancement processing. Subsequent subsections will define the procedures in more detail.

4.1.1 Types of Keys

A two-level keying hierarchy is used to support privacy-enhanced message transmission:

1. Data Encrypting Keys (DEKs) are used for encryption of message text and (with certain choices among a set of alternative algorithms) for computation of message integrity

check (MIC) quantities. DEKs are generated individually for each transmitted message; no predistribution of DEKs is needed to support privacy-enhanced message transmission.

2. Interchange Keys (IKs) are used to encrypt DEKs for transmission within messages. Ordinarily, the same IK will be used for all messages sent from a given originator to a given recipient over a period of time. Each transmitted message includes a representation of the DEK(s) used for message encryption and/or MIC computation, encrypted under an individual IK per named recipient. The representation is associated with "X-Sender-ID:" and "X-Recipient-ID:" fields, which allow each individual recipient to identify the IK used to encrypt DEKs and/or MICs for that recipient's use. Given an appropriate IK, a recipient can decrypt the corresponding transmitted DEK representation, yielding the DEK required for message text decryption and/or MIC verification. The definition of an IK differs depending on whether symmetric or asymmetric cryptography is used for DEK encryption:

Linn

[Page 7]

RFC 1113

Mail Privacy: Procedures

August 1989

- 2a. When symmetric cryptography is used for DEK encryption, an IK is a single symmetric key shared between an originator and a recipient. In this case, the same IK is used to encrypt MICs as well as DEKs for transmission. Version/expiration information and IA identification associated with the originator and with the recipient must be concatenated in order to fully qualify a symmetric IK.
- 2b. When asymmetric cryptography is used, the IK component used for DEK encryption is the public component of the recipient. The IK component used for MIC encryption is the private component of the originator, and therefore only one encrypted MIC representation need be included per message, rather than one per recipient. Each of these IK components can be fully qualified in an

"X-Recipient-ID:" or "X-Sender-ID:" field,
respectively.

4.1.2 Processing Procedures

When privacy enhancement processing is to be performed on an outgoing message, a DEK is generated [1] for use in message encryption and (if a chosen MIC algorithm requires a key) a variant of the DEK is formed for use in MIC computation. DEK generation can be omitted for the case of a message in which all contents are excluded from encryption, unless a chosen MIC computation algorithm requires a DEK.

An "X-Sender-ID:" field is included in the header to provide one identification component for the IK(s) used for message processing. IK components are selected for each individually named recipient; a corresponding "X-Recipient-ID:" field, interpreted in the context of a prior "X-Sender-ID:" field, serves to identify each IK. Each "X-Recipient-ID:" field is followed by an "X-Key-Info:" field, which transfers a DEK encrypted under the IK appropriate for the specified recipient. When symmetric key management is used for a given recipient, the "X-Key-Info:" field also transfers the message's computed MIC, encrypted under the recipient's IK. When asymmetric key management is used, a prior "X-MIC-Info:" field carries the

message's MIC encrypted under the private component of the sender.

A four-phase transformation procedure is employed in order to represent encrypted message text in a universally transmissible form and to enable messages encrypted on one type of host computer to be decrypted on a different type of host computer. A plaintext message is accepted in local form, using the host's native character set and

Linn

[Page 8]

RFC 1113

Mail Privacy: Procedures

August 1989

line representation. The local form is converted to a canonical message text representation, defined as equivalent to the inter-SMTP representation of message text. This canonical representation forms the input to the MIC computation and encryption processes.

For encryption purposes, the canonical representation is padded as required by the encryption algorithm. The padded canonical representation is encrypted (except for any regions which are explicitly excluded from encryption). The encrypted text (along with

the canonical representation of regions which were excluded from encryption) is encoded into a printable form. The printable form is composed of a restricted character set which is chosen to be universally representable across sites, and which will not be disrupted by processing within and between MTS entities.

The output of the encoding procedure is combined with a set of header fields carrying cryptographic control information. The result is passed to the electronic mail system to be encapsulated as the text portion of a transmitted message.

When a privacy-enhanced message is received, the cryptographic control fields within its text portion provide the information required for the authorized recipient to perform MIC verification and decryption of the received message text. First, the printable encoding is converted to a bitstring. Encrypted portions of the transmitted message are decrypted. The MIC is verified. The canonical representation is converted to the recipient's local form, which need not be the same as the sender's local form.

4.2 Encryption Algorithms and Modes

For purposes of this RFC, the Block Cipher Algorithm DEA-1, defined

in ANSI X3.92-1981 [2] shall be used for encryption of message text. The DEA-1 is equivalent to the Data Encryption Standard (DES), as defined in FIPS PUB 46 [3]. When used for encryption of text, the DEA-1 shall be used in the Cipher Block Chaining (CBC) mode, as defined in ISO IS 8372 [4]. The identifier string "DES-CBC", defined in RFC-1115, signifies this algorithm/mode combination. The CBC mode definition in IS 8372 is equivalent to that provided in FIPS PUB 81 [5] and in ANSI X3.106-1983 [16]. Use of other algorithms and/or modes for message text processing will require case-by-case study to determine applicability and constraints. Additional algorithms and modes approved for use in this context will be specified in successors to RFC-1115.

It is an originator's responsibility to generate a new pseudorandom initializing vector (IV) for each privacy-enhanced electronic mail message unless the entirety of the message is excluded from

encryption. Section 4.3.1 of [17] provides rationale for this requirement, even in a context where individual DEKs are generated for individual messages. The IV will be transmitted with the message.

Certain operations require that one key be encrypted under an interchange key (IK) for purposes of transmission. A header facility indicates the mode in which the IK is used for encryption. RFC-1115 specifies encryption algorithm/mode identifiers, including DES-ECB, DES-EDE, and RSA. All implementations using symmetric key management should support DES-ECB IK use, and all implementations using asymmetric key management should support RSA IK use.

RFC-1114, released concurrently with this RFC, specifies asymmetric, certificate-based key management procedures to support the message processing procedures defined in this document. The message processing procedures can also be used with symmetric key management, given prior distribution of suitable symmetric IKs through out-of-band means. Support for the asymmetric approach defined in RFC-1114 is strongly recommended.

4.3 Privacy Enhancement Message Transformations

4.3.1 Constraints

An electronic mail encryption mechanism must be compatible with the transparency constraints of its underlying electronic mail facilities. These constraints are generally established based on expected user requirements and on the characteristics of anticipated endpoint and transport facilities. An encryption mechanism must also be compatible with the local conventions of the computer systems which it interconnects. In our approach, a canonicalization step is performed to abstract out local conventions and a subsequent encoding step is performed to conform to the characteristics of the underlying mail transport medium (SMTP). The encoding conforms to SMTP constraints, established to support interpersonal messaging. SMTP's rules are also used independently in the canonicalization process. RFC-821's [7] Section 4.5 details SMTP's transparency constraints.

To prepare a message for SMTP transmission, the following requirements must be met:

1. All characters must be members of the 7-bit ASCII character set.

2. Text lines, delimited by the character pair <CR><LF>, must be no more than 1000 characters long.

Linn

[Page 10]

RFC 1113

Mail Privacy: Procedures

August 1989

3. Since the string <CR><LF>.<CR><LF> indicates the end of a message, it must not occur in text prior to the end of a message.

Although SMTP specifies a standard representation for line delimiters (ASCII <CR><LF>), numerous systems use a different native representation to delimit lines. For example, the <CR><LF> sequences delimiting lines in mail inbound to UNIX systems are transformed to single <LF>s as mail is written into local mailbox files. Lines in mail incoming to record-oriented systems (such as VAX VMS) may be converted to appropriate records by the destination SMTP [8] server. As a result, if the encryption process generated <CR>s or <LF>s, those characters might not be accessible to a recipient UA program at

a destination which uses different line delimiting conventions. It is also possible that conversion between tabs and spaces may be performed in the course of mapping between inter-SMTP and local format; this is a matter of local option. If such transformations changed the form of transmitted ciphertext, decryption would fail to regenerate the transmitted plaintext, and a transmitted MIC would fail to compare with that computed at the destination.

The conversion performed by an SMTP server at a system with EBCDIC as a native character set has even more severe impact, since the conversion from EBCDIC into ASCII is an information-losing transformation. In principle, the transformation function mapping between inter-SMTP canonical ASCII message representation and local format could be moved from the SMTP server up to the UA, given a means to direct that the SMTP server should no longer perform that transformation. This approach has a major disadvantage: internal file (e.g., mailbox) formats would be incompatible with the native forms used on the systems where they reside. Further, it would require modification to SMTP servers, as mail would be passed to SMTP in a different representation than it is passed at present.

4.3.2 Approach

Our approach to supporting privacy-enhanced mail across an environment in which intermediate conversions may occur encodes mail in a fashion which is uniformly representable across the set of privacy-enhanced UAs regardless of their systems' native character sets. This encoded form is used to represent mail text from sender to recipient, but the encoding is not applied to enclosing mail transport headers or to encapsulated headers inserted to carry control information between privacy-enhanced UAs. The encoding's characteristics are such that the transformations anticipated between sender and recipient UAs will not prevent an encoded message from being decoded properly at its destination.

Linn

[Page 11]

RFC 1113

Mail Privacy: Procedures

August 1989

A sender may exclude one or more portions of a message from encryption processing, but authentication processing is always applied to the entirety of message text. Explicit action is required to exclude a portion of a message from encryption processing; by

default, encryption is applied to the entirety of message text. The user-level delimiter which specifies such exclusion is a local matter, and hence may vary between sender and recipient, but all systems should provide a means for unambiguous identification of areas excluded from encryption processing.

An outbound privacy-enhanced message undergoes four transformation steps, described in the following four subsections.

4.3.2.1 Step 1: Local Form

The message text is created in the system's native character set, with lines delimited in accordance with local convention.

4.3.2.2 Step 2: Canonical Form

The entire message text, including both those portions subject to encipherment processing and those portions excluded from such processing, is converted to a universal canonical form, analogous to the inter-SMTP representation [9] as defined in RFC-821 and RFC-822 [10] (ASCII character set, <CR><LF> line delimiters). The processing required to perform this conversion is minimal on systems whose native character set is ASCII. (Note: Since the output of the

canonical encoding process will never be submitted directly to SMTP, but only to subsequent steps of the privacy enhancement encoding process, the dot-stuffing transformation discussed in RFC-821, section 4.5.2, is not required.) Since a message is converted to a standard character set and representation before encryption, it can be decrypted and its MIC can be verified at any type of destination host computer. The decryption and MIC verification is performed before any conversions which may be necessary to transform the message into a destination-specific local form.

4.3.2.3 Step 3: Authentication and Encipherment

The canonical form is input to the selected MIC computation algorithm in order to compute an integrity check quantity for the message. No padding is added to the canonical form before submission to the MIC computation algorithm, although certain MIC algorithms will apply their own padding in the course of computing a MIC.

Padding is applied to the canonical form as needed to perform encryption in the DEA-1 CBC mode, as follows: The number of octets to be encrypted is determined by subtracting the number of octets

Linn

[Page 12]

RFC 1113

Mail Privacy: Procedures

August 1989

excluded from encryption from the total length of the canonically encoded text. Octets with the hexadecimal value FF (all ones) are appended to the canonical form as needed so that the text octets to be encrypted, along with the added padding octets, fill an integral number of 8-octet encryption quanta. No padding is applied if the number of octets to be encrypted is already an integral multiple of 8. The use of hexadecimal FF (a value outside the 7-bit ASCII set) as a padding value allows padding octets to be distinguished from valid data without inclusion of an explicit padding count indicator.

The regions of the message which have not been excluded from encryption are encrypted. To support selective encipherment processing, an implementation must retain internal indications of the positions of excluded areas excluded from encryption with relation to non-excluded areas, so that those areas can be properly delimited in the encoding procedure defined in step 4. If a region excluded from encryption intervenes between encrypted regions, cryptographic state (e.g., IVs and accumulation of octets into encryption quanta) is

preserved and continued after the excluded region.

4.3.2.4 Step 4: Printable Encoding

Proceeding from left to right, the bit string resulting from step 3 is encoded into characters which are universally representable at all sites, though not necessarily with the same bit patterns (e.g., although the character "E" is represented in an ASCII-based system as hexadecimal 45 and as hexadecimal C5 in an EBCDIC-based system, the local significance of the two representations is equivalent). This encoding step is performed for all privacy-enhanced messages, even if an entire message is excluded from encryption.

A 64-character subset of International Alphabet IA5 is used, enabling 6 bits to be represented per printable character. (The proposed subset of characters is represented identically in IA5 and ASCII.) Two additional characters, "=" and "*", are used to signify special processing functions. The character "=" is used for padding within the printable encoding procedure. The character "*" is used to delimit the beginning and end of a region which has been excluded from encipherment processing. The encoding function's output is delimited into text lines (using local conventions), with each line except the last containing exactly 64 printable characters and the

final line containing 64 or fewer printable characters. (This line length is easily printable and is guaranteed to satisfy SMTP's 1000-character transmitted line length limit.)

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right across a 24-bit input group extracted from the output of step 3, each 6-bit

Linn

[Page 13]

RFC 1113

Mail Privacy: Procedures

August 1989

group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 0, are selected so as to be universally representable, and the set excludes characters with particular significance to SMTP (e.g., ".", "<CR>", "<LF>").

Special processing is performed if fewer than 24 bits are available in an input group, either at the end of a message or (when the selective encryption facility is invoked) at the end of an encrypted

region or an excluded region. A full encoding quantum is always completed at the end of a message and before the delimiter "*" is output to initiate or terminate the representation of a block excluded from encryption. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Output character positions which are not required to represent actual input data are set to the character "=". Since all canonically encoded output is an integral number of octets, only the following cases can arise: (1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding, (2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or (3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

Linn

[Page 14]

RFC 1113

Mail Privacy: Procedures

August 1989

4.3.2.5 Summary of Transformations

In summary, the outbound message is subjected to the following composition of transformations:

```
Transmit_Form = Encode(Encipher(Canonicalize(Local_Form)))
```

The inverse transformations are performed, in reverse order, to process inbound privacy-enhanced mail:

```
Local_Form = DeCanonicalize(Decipher(Decode(Transmit_Form)))
```

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7

9 J	26 a	43 r	60 8
10 K	27 b	44 s	61 9
11 L	28 c	45 t	62 +
12 M	29 d	46 u	63 /
13 N	30 e	47 v	
14 O	31 f	48 w	(pad) =
15 P	32 g	49 x	
16 Q	33 h	50 y	(1) *

(1) The character "*" is used to enclose portions of an encoded message to which encryption processing has not been applied.

Printable Encoding Characters

Table 1

Note that the local form and the functions to transform messages to and from canonical form may vary between the sender and recipient systems without loss of information.

4 4 Encapsulation Mechanism

Encapsulation of privacy-enhanced messages within an enclosing layer

Linn

[Page 15]

RFC 1113

Mail Privacy: Procedures

August 1989

of headers interpreted by the electronic mail transport system offers a number of advantages in comparison to a flat approach in which certain fields within a single header are encrypted and/or carry cryptographic control information. Encapsulation provides generality and segregates fields with user-to-user significance from those transformed in transit. All fields inserted in the course of encryption/authentication processing are placed in the encapsulated header. This facilitates compatibility with mail handling programs which accept only text, not header fields, from input files or from other programs. Further, privacy enhancement processing can be applied recursively. As far as the MTS is concerned, information incorporated into cryptographic authentication or encryption processing will reside in a message's text portion, not its header portion.

The encapsulation mechanism to be used for privacy-enhanced mail is derived from that described in RFC-934 [11] which is, in turn, based on precedents in the processing of message digests in the Internet community. To prepare a user message for encrypted or authenticated transmission, it will be transformed into the representation shown in Figure 1.

As a general design principle, sensitive data is protected by incorporating the data within the encapsulated text rather than by applying measures selectively to fields in the enclosing header. Examples of potentially sensitive header information may include fields such as "Subject:", with contents which are significant on an end-to-end, inter-user basis. The (possibly empty) set of headers to which protection is to be applied is a user option. It is strongly recommended, however, that all implementations should replicate copies of "X-Sender-ID:" and "X-Recipient-ID:" fields within the encapsulated text.

If a user wishes disclosure protection for header fields, they must occur only in the encapsulated text and not in the enclosing or encapsulated header. If disclosure protection is desired for a message's subject indication, it is recommended that the enclosing

header contain a "Subject:" field indicating that "Encrypted Mail Follows".

If an authenticated version of header information is desired, that data can be replicated within the encapsulated text portion in addition to its inclusion in the enclosing header. For example, a sender wishing to provide recipients with a protected indication of a message's position in a series of messages could include a copy of a timestamp or message counter field within the encapsulated text.

A specific point regarding the integration of privacy-enhanced mail

Linn

[Page 16]

RFC 1113

Mail Privacy: Procedures

August 1989

facilities with the message encapsulation mechanism is worthy of note. The subset of IA5 selected for transmission encoding intentionally excludes the character "-", so encapsulated text can be distinguished unambiguously from a message's closing encapsulation boundary (Post-EB) without recourse to character stuffing.

Enclosing Header Portion

(Contains header fields per RFC-822)

Blank Line

(Separates Enclosing Header from Encapsulated Message)

Encapsulated Message

Pre-Encapsulation Boundary (Pre-EB)

-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----

Encapsulated Header Portion

(Contains encryption control fields inserted in plaintext.

Examples include "X-DEK-Info:", "X-Sender-ID:", and

"X-Key-Info:".

Note that, although these control fields have line-oriented representations similar to RFC-822 header fields, the set of fields valid in this context is disjoint from those used in RFC-822 processing.)

Blank Line

(Separates Encapsulated Header from subsequent encoded

Encapsulated Text Portion)

Encapsulated Text Portion

(Contains message data encoded as specified in Section 4.3;
may incorporate protected copies of enclosing and
encapsulated header fields such as "Subject:", etc.)

Post-Encapsulation Boundary (Post-EB)

-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----

Message Encapsulation

Figure 1

4.5 Mail for Mailing Lists

When mail is addressed to mailing lists, two different methods of processing can be applicable: the IK-per-list method and the IK-per-recipient method. The choice depends on the information available to

Linn

[Page 17]

RFC 1113

Mail Privacy: Procedures

August 1989

the sender and on the sender's preference.

If a message's sender addresses a message to a list name or alias, use of an IK associated with that name or alias as a entity (IK-per-list), rather than resolution of the name or alias to its constituent destinations, is implied. Such an IK must, therefore, be available to all list members. For the case of asymmetric key management, the list's private component must be available to all list members. This alternative will be the normal case for messages sent via remote exploder sites, as a sender to such lists may not be cognizant of the set of individual recipients. Unfortunately, it implies an undesirable level of exposure for the shared IK, and makes its revocation difficult. Moreover, use of the IK-per-list method allows any holder of the list's IK to masquerade as another sender to the list for authentication purposes.

If, in contrast, a message's sender is equipped to expand the destination mailing list into its individual constituents and elects to do so (IK-per-recipient), the message's DEK (and, in the symmetric

key management case, MIC) will be encrypted under each per-recipient IK and all such encrypted representations will be incorporated into the transmitted message. Note that per-recipient encryption is required only for the relatively small DEK and MIC quantities carried in the "X-Key-Info:" field, not for the message text which is, in general, much larger. Although more IKs are involved in processing under the IK-per-recipient method, the pairwise IKs can be individually revoked and possession of one IK does not enable a successful masquerade of another user on the list.

4.6 Summary of Encapsulated Header Fields

This section summarizes the syntax and semantics of the encapsulated header fields to be added to messages in the course of privacy enhancement processing. The fields are presented in three groups. Normally, the groups will appear in encapsulated headers in the order in which they are shown, though not all fields in each group will appear in all messages. In certain indicated cases, it is recommended that the fields be replicated within the encapsulated text portion as well as being included within the encapsulated header. Figures 2 and 3 show the appearance of small example encapsulated messages. Figure 2 assumes the use of symmetric cryptography for key management. Figure 3 illustrates an example encapsulated message in which

asymmetric key management is used.

Unless otherwise specified, all field arguments are processed in a case-sensitive fashion. In most cases, numeric quantities are represented in header fields as contiguous strings of hexadecimal digits, where each digit is represented by a character from the

Linn

[Page 18]

RFC 13

Mail Privacy: Procedures

August 1989

ranges "0"-"9" or upper case "A"-"F". Since public-key certificates and quantities encrypted using asymmetric algorithms are large in size, use of a more space-efficient encoding technique is appropriate for such quantities, and the encoding mechanism defined in Section 4.3.2.4 of this RFC, representing 6 bits per printed character, is adopted. The example shown in Figure 3 shows asymmetrically encrypted quantities (e.g., "X-MIC-Info:", "X-Key-Info:") with 64-character printed representations, corresponding to 384 bits. The fields carrying asymmetrically encrypted quantities also illustrate

the use of folding as defined in RFC-822, section 3.1.1.

-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----

X-Proc-Type: 3, ENCRYPTED

X-DEK-Info: DES-CBC, F8143EDE5960C597

X-Sender-ID: linn@ccy.bbn.com::

X-Recipient-ID: linn@ccy.bbn.com:ptf-kmc:3

X-Key-Info: DES-ECB, RSA-MD2, 9FD3AAD2F2691B9A, B70665BB9BF7CBCD,
A60195DB94F727D3

X-Recipient-ID: privacy-tf@venera.isi.edu:ptf-kmc:4

X-Key-Info: DES-ECB, RSA-MD2, 161A3F75DC82EF26, E2EF532C65CBCFF7,
9F83A2658132DB47

LLrHB0eJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoH1nvNSIWL9M
8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHU1BLpvXROUrUzYbkNpkOagV2IzUpk
J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcMlK1Z6720dcBWGGsDLpTpSCnpot
dXd/H5LMDWnonNvPCwQUHt==

-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----

Example Encapsulated Message (Symmetric Case)

Figure 2

-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----

X-Proc-Type: 3, ENCRYPTED

X-DEK-Info: DES-CBC,F8143EDE5960C597

X-Sender-ID: linn@ccy.bbn.com::

X-Certificate:

jHU1BLpvXR0UrUzYbkNpkOagV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIk
YbkNpkOagV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHU1BLpvXR0UrUz
agV2IzUpk8tEjmFjHU1BLpvXR0UrUz/zxB+bATMtPjCUWbz8Lr9wloXIkYbkNpkO

X-Issuer-Certificate:

TMtPjCUWbz8Lr9wloXIkybkNpkOagV2IzUpk8tEjmFjHU1BLpvXR0UUrUz/zxB+bA
IkjHU1BLpvXR0UUrUzYbkNpkOagV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloX
vXR0UUrUzYbkNpkOagV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkJHU1BLp

X-MIC-Info: RSA-MD2,RSA,

5rDqUcMlK1Z6720dcBWGGsDLpTpSCnpotJ6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz

X-Recipient-ID: linn@ccy.bbn.com:RSADSI:3

Linn

[Page 19]

X-Key-Info: RSA,

1BLpvXROUrUzYbkNpk0agV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkJHU

X-Recipient-ID: privacy-tf@venera.isi.edu:RSADSI:4

X-Key-Info: RSA,

NcUk2jHEUSoH1nvNSIWL9MLLrHBOeJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72oh

LLrHBOeJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoH1nvNSIWL9M

8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkJHU1BLpvXROUrUzYbkNpk0agV2IzUpk

J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcMlK1Z6720dcBWGGsDLpTpSCnpot

dXd/H5LMDWnonNvPCwQUHt==

-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----

Example Encapsulated Message (Asymmetric Case)

Figure 3

Although the encapsulated header fields resemble RFC-822 header fields, they are a disjoint set and will not in general be processed by the same parser which operates on enclosing header fields. The complexity of lexical analysis needed and appropriate for encapsulated header field processing is significantly less than that appropriate to RFC-822 header processing. For example, many characters with special significance to RFC-822 at the syntactic

level have no such significance within encapsulated header fields.

When the length of an encapsulated header field is longer than the size conveniently printable on a line, whitespace may be used to fold the field in the manner of RFC-822, section 3.1.1. Any such inserted whitespace is not to be interpreted as a part of a subfield. As a particular example, due to the length of public-key certificates and of quantities encrypted using asymmetric algorithms, such quantities may often need to be folded across multiple printed lines. In order to facilitate such folding in a uniform manner, the bits representing such a quantity are to be divided into an ordered set (with leftmost bits first) of zero or more 384-bit groups (corresponding to 64-character printed representations), followed by a final group of bits which may be any length up to 384 bits.

4.6.1 Per-Message Encapsulated Header Fields

This group of encapsulated header fields contains fields which occur no more than once in a privacy-enhanced message, generally preceding all other encapsulated header fields.

4.6.1.1 X-Proc-Type Field

The "X-Proc-Type:" encapsulated header field, required for all privacy-enhanced messages, identifies the type of processing

Linn

[Page 20]

RFC 1113

Mail Privacy: Procedures

August 1989

performed on the transmitted message. Only one "X-Proc-Type:" field occurs in a message; the "X-Proc-Type:" field must be the first encapsulated header field in the message.

The "X-Proc-Type:" field has two subfields, separated by a comma. The first subfield is a decimal number which is used to distinguish among incompatible encapsulated header field interpretations which may arise as changes are made to this standard. Messages processed according to this RFC will carry the subfield value "3" to distinguish them from messages processed in accordance with prior RFCs 989 and 1040.

The second subfield may assume one of two string values: "ENCRYPTED" or "MIC-ONLY". Unless all of a message's encapsulated text is

excluded from encryption, the "X-Proc-Type:" field's second subfield must specify "ENCRYPTED". Specification of "MIC-ONLY", when applied in conjunction with certain combinations of key management and MIC algorithm options, permits certain fields which are superfluous in the absence of encryption to be omitted from the encapsulated header. In particular, "X-Recipient-ID:" and "X-Key-Info:" fields can be omitted for recipients for whom asymmetric cryptography is used, assuming concurrent use of a keyless MIC computation algorithm. The "X-DEK-Info:" field can be omitted for all "MIC-ONLY" messages.

4.6.1.2 X-DEK-Info Field

The "X-DEK-Info:" encapsulated header field identifies the message text encryption algorithm and mode, and also carries the Initializing Vector used for message encryption. No more than one "X-DEK-Info:" field occurs in a message; the field is required except for messages specified as "MIC-ONLY" in the "X-Proc-Type:" field.

The "X-DEK-Info:" field carries two arguments, separated by a comma. For purposes of this RFC, the first argument must be the string "DES-CBC", signifying (as defined in RFC-1115) use of the DES algorithm in the CBC mode. The second argument represents a 64-bit Initializing Vector (IV) as a contiguous string of 16 hexadecimal

digits. Subsequent revisions of RFC-1115 will specify any additional values which may appear as the first argument of this field.

4.6.2 Encapsulated Header Fields Normally Per-Message

This group of encapsulated header fields contains fields which ordinarily occur no more than once per message. Depending on the key management option(s) employed, some of these fields may be absent from some messages. The "X-Sender-ID" field may occur more than once in a message if different sender-oriented IK components (perhaps corresponding to different versions) must be used for different

Linn

[Page 21]

RFC 1113

Mail Privacy: Procedures

August 1989

recipients. In this case later occurrences override prior occurrences. If a mixture of symmetric and asymmetric key distribution is used within a single message, the recipients for each type of key distribution technology should be grouped together to simplify parsing.

4.6.2.1 X-Sender-ID Field

The "X-Sender-ID:" encapsulated header field, required for all privacy-enhanced messages, identifies a message's sender and provides the sender's IK identification component. It should be replicated within the encapsulated text. The IK identification component carried in an "X-Sender-ID:" field is used in conjunction with all subsequent "X-Recipient-ID:" fields until another "X-Sender-ID:" field occurs; the ordinary case will be that only a single "X-Sender-ID:" field will occur, prior to any "X-Recipient-ID:" fields.

The "X-Sender-ID:" field contains (in order) an Entity Identifier subfield, an (optional) Issuing Authority subfield, and an (optional) Version/Expiration subfield. The optional subfields are omitted if their use is rendered redundant by information carried in subsequent "X-Recipient-ID:" fields; this will ordinarily be the case where symmetric cryptography is used for key management. The subfields are delimited by the colon character (":"), optionally followed by whitespace.

Section 5.2, Interchange Keys, discusses the semantics of these subfields and specifies the alphabet from which they are chosen.

Note that multiple "X-Sender-ID:" fields may occur within a single encapsulated header. All "X-Recipient-ID:" fields are interpreted in the context of the most recent preceding "X-Sender-ID:" field; it is illegal for an "X-Recipient-ID:" field to occur in a header before an "X-Sender-ID:" has been provided.

4.6.2.2 X-Certificate Field

The "X-Certificate:" encapsulated header field is used only when asymmetric key management is employed for one or more of a message's recipients. To facilitate processing by recipients (at least in advance of general directory server availability), inclusion of this field in all messages is strongly recommended. The field transfers a sender's certificate as a numeric quantity, represented with the encoding mechanism defined in Section 4.3.2.4 of this RFC. The semantics of a certificate are discussed in RFC-1114. The certificate carried in an "X-Certificate:" field is used in conjunction with "X-Sender-ID:" and "X-Recipient-ID:" fields for which asymmetric key management is employed.

Linn

[Page 22]

RFC 1113

Mail Privacy: Procedures

August 1989

4.6.2.3 X-MIC-Info Field

The "X-MIC-Info:" encapsulated header field, used only when asymmetric key management is employed for at least one recipient of a message, carries three arguments, separated by commas. The first argument identifies the algorithm under which the accompanying MIC is computed; RFC-1115 specifies the acceptable set of MIC algorithm identifiers. The second argument identifies the algorithm under which the accompanying MIC is encrypted; for purposes of this RFC, the string "RSA" as described in RFC-1115 must occur, identifying use of the RSA algorithm. The third argument is a MIC, asymmetrically encrypted using the originator's private component. As discussed earlier in this section, the asymmetrically encrypted MIC is represented using the technique described in Section 4.3.2.4 of this RFC.

The "X-MIC-Info:" field will occur immediately following the message's "X-Sender-ID:" field and any "X-Certificate:" or "X-Issuer-Certificate:" fields. Analogous to the "X-Sender-ID:" field,

an "X-MIC-Info:" field applies to all subsequent recipients for whom asymmetric key management is used.

4.6.3 Encapsulated Header Fields with Variable Occurrences

This group of encapsulated header fields contains fields which will normally occur variable numbers of times within a message, with numbers of occurrences ranging from zero to non-zero values which are independent of the number of recipients.

4.6.3.1 X-Issuer-Certificate Field

The "X-Issuer-Certificate:" encapsulated header field is meaningful only when asymmetric key management is used for at least one of a message's recipients. A typical "X-Issuer-Certificate:" field would contain the certificate containing the public component used to sign the certificate carried in the message's "X-Certificate:" field, for recipients' use in chaining through that certificate's certification path. Other "X-Issuer-Certificate:" fields, typically representing higher points in a certification path, also may be included by a sender. The order in which "X-Issuer-Certificate:" fields are included need not correspond to the order of the certification path; the order of that path may in general differ from the viewpoint of

different recipients. More information on certification paths can be found in RFC-1114.

The certificate is represented in the same manner as defined for the "X-Certificate:" field, and any "X-Issuer-Certificate:" fields will ordinarily follow the "X-Certificate:" field directly. Use of the

Linn

[Page 23]

RFC 1113

Mail Privacy: Procedures

August 1989

"X-Issuer-Certificate:" field is optional even when asymmetric key management is employed, although its incorporation is strongly recommended in the absence of alternate directory server facilities from which recipients can access issuers' certificates.

4.6.4 Per-Recipient Encapsulated Header Fields

This group of encapsulated header fields normally appears once for each of a message's named recipients. As a special case, these fields may be omitted in the case of a "MIC-ONLY" message to

recipients for whom asymmetric key management is employed, given that the chosen MIC algorithm is keyless.

4.6.4.1 X-Recipient-ID Field

The "X-Recipient-ID:" encapsulated header field identifies a recipient and provides the recipient's IK identification component. One "X-Recipient-ID:" field is included for each of a message's named recipients. It should be replicated within the encapsulated text. The field contains (in order) an Entity Identifier subfield, an Issuing Authority subfield, and a Version/Expiration subfield. The subfields are delimited by the colon character (":"), optionally followed by whitespace.

Section 5.2, Interchange Keys, discusses the semantics of the subfields and specifies the alphabet from which they are chosen. All "X-Recipient-ID:" fields are interpreted in the context of the most recent preceding "X-Sender-ID:" field; it is illegal for an "X-Recipient-ID:" field to occur in a header before an "X-Sender-ID:" has been provided.

4.6.4.2 X-Key-Info Field

One "X-Key-Info:" field is included for each of a message's named recipients. Each "X-Key-Info:" field is interpreted in the context of the most recent preceding "X-Recipient-ID:" field; normally, an "X-Key-Info:" field will immediately follow its associated "X-Recipient-ID:" field. The field's argument(s) differ depending on whether symmetric or asymmetric key management is used for a particular recipient.

4.6.4.2.1 Symmetric Key Management

When symmetric key management is employed for a given recipient, the "X-Key-Info:" encapsulated header field transfers four items, separated by commas: an IK Use Indicator, a MIC Algorithm Indicator, a DEK and a MIC. The IK Use Indicator identifies the algorithm and mode in which the identified IK was used for DEK encryption for a

particular recipient. For recipients for whom symmetric key

management is used, it may assume the reserved string values "DES-ECB" or "DES-EDE", as defined in RFC-1115.

The MIC Algorithm Indicator identifies the MIC computation algorithm used for a particular recipient; values for this subfield are defined in RFC-1115. The DEK and MIC are encrypted under the IK identified by a preceding "X-Recipient-ID:" field and prior "X-Sender-ID:" field; they are represented as two strings of contiguous hexadecimal digits, separated by a comma.

When DEA-1 is used for message text encryption, the DEK representation will be 16 hexadecimal digits (corresponding to a 64-bit key); this subfield can be extended to 32 hexadecimal digits (corresponding to a 128-bit key) if required to support other algorithms.

Symmetric encryption of MICs is always performed in the same encryption mode used to encrypt the message's DEK. Encrypted MICs, like encrypted DEKs, are represented as contiguous strings of hexadecimal digits. The size of a MIC is dependent on the choice of MIC algorithm as specified in the MIC Algorithm Indicator subfield.

4.6.4.2.2 Asymmetric Key Management

When asymmetric key management is employed for a given recipient, the "X-Key-Info:" field transfers two quantities, separated by commas. The first argument is an IK Use Indicator identifying the algorithm (and mode, if applicable) in which the DEK is encrypted; for purposes of this RFC, the IK Use Indicator subfield will always assume the reserved string value "RSA" (as defined in RFC-1115) for recipients for whom asymmetric key management is employed, signifying use of the RSA algorithm. The second argument is a DEK, encrypted (using asymmetric encryption) under the recipient's public component.

Throughout this RFC we have adopted the terms "private component" and "public component" to refer to the quantities which are, respectively, kept secret and made publically available in asymmetric cryptosystems. This convention is adopted to avoid possible confusion arising from use of the term "secret key" to refer to either the former quantity or to a key in a symmetric cryptosystem.

As discussed earlier in this section, the asymmetrically encrypted DEK is represented using the technique described in Section 4.3.2.4 of this RFC.

Linn

[Page 25]

RFC 1113

Mail Privacy: Procedures

August 1989

5. Key Management

Several cryptographic constructs are involved in supporting the privacy-enhanced message processing procedure. A set of fundamental elements is assumed. Data Encrypting Keys (DEKs) are used to encrypt message text and (for some MIC computation algorithms) in the message integrity check (MIC) computation process. Interchange Keys (IKs) are used to encrypt DEKs and MICs for transmission with messages. In a certificate-based asymmetric key management architecture, certificates are used as a means to provide entities' public components and other information in a fashion which is securely bound by a central authority. The remainder of this section provides more information about these constructs.

5.1 Data Encrypting Keys (DEKs)

Data Encrypting Keys (DEKs) are used for encryption of message text and (with some MIC computation algorithms) for computation of message integrity check quantities (MICs). It is strongly recommended that DEKs be generated and used on a one-time, per-message, basis. A transmitted message will incorporate a representation of the DEK encrypted under an appropriate interchange key (IK) for each of the named recipients.

DEK generation can be performed either centrally by key distribution centers (KDCs) or by endpoint systems. Dedicated KDC systems may be able to implement stronger algorithms for random DEK generation than can be supported in endpoint systems. On the other hand, decentralization allows endpoints to be relatively self-sufficient, reducing the level of trust which must be placed in components other than a message's originator and recipient. Moreover, decentralized DEK generation at endpoints reduces the frequency with which senders must make real-time queries of (potentially unique) servers in order to send mail, enhancing communications availability.

When symmetric cryptography is used, one advantage of centralized KDC-based generation is that DEKs can be returned to endpoints already encrypted under the IKs of message recipients rather than

providing the IKs to the senders. This reduces IK exposure and simplifies endpoint key management requirements. This approach has less value if asymmetric cryptography is used for key management, since per-recipient public IK components are assumed to be generally available and per-sender private IK components need not necessarily be shared with a KDC.

5.2 Interchange Keys (IKs)

Interchange Key (IK) components are used to encrypt DEKs and MICs.

Linn

[Page 26]

RFC 1113

Mail Privacy: Procedures

August 1989

In general, IK granularity is at the pairwise per-user level except for mail sent to address lists comprising multiple users. In order for two principals to engage in a useful exchange of privacy-enhanced electronic mail using conventional cryptography, they must first possess common IK components (when symmetric key management is used) or complementary IK components (when asymmetric key management is

used). When symmetric cryptography is used, the IK consists of a single component, used to encrypt both DEKs and MICs. When asymmetric cryptography is used, a recipient's public component is used as an IK to encrypt DEKs (a transformation invertible only by a recipient possessing the corresponding private component), and the originator's private component is used to encrypt MICs (a transformation invertible by all recipients, since the originator's certificate provides the necessary public component of the originator).

While this RFC does not prescribe the means by which interchange keys are provided to appropriate parties, it is useful to note that such means may be centralized (e.g., via key management servers) or decentralized (e.g., via pairwise agreement and direct distribution among users). In any case, any given IK component is associated with a responsible Issuing Authority (IA). When certificate-based asymmetric key management, as discussed in RFC-1114, is employed, the IA function is performed by a Certification Authority (CA).

When an IA generates and distributes an IK component, associated control information is provided to direct how it is to be used. In order to select the appropriate IK(s) to use in message encryption, a sender must retain a correspondence between IK components and the

recipients with which they are associated. Expiration date information must also be retained, in order that cached entries may be invalidated and replaced as appropriate.

Since a message may be sent with multiple IK components identified, corresponding to multiple intended recipients, each recipient's UA must be able to determine that recipient's intended IK component. Moreover, if no corresponding IK component is available in the recipient's database when a message arrives, the recipient must be able to identify the required IK component and identify that IK component's associated IA. Note that different IKs may be used for different messages between a pair of communicants. Consider, for example, one message sent from A to B and another message sent (using the IK-per-list method) from A to a mailing list of which B is a member. The first message would use IK components associated individually with A and B, but the second would use an IK component shared among list members.

When a privacy-enhanced message is transmitted, an indication of the

IK components used for DEK and MIC encryption must be included. To this end, the "X-Sender-ID:" and "X-Recipient-ID:" encapsulated header fields provide the following data:

1. Identification of the relevant Issuing Authority (IA subfield)
2. Identification of an entity with which a particular IK component is associated (Entity Identifier or EI subfield)
3. Version/Expiration subfield

The colon character (":") is used to delimit the subfields within an "X-Sender-ID:" or "X-Recipient-ID:". The IA, EI, and version/expiration subfields are generated from a restricted character set, as prescribed by the following BNF (using notation as defined in RFC-822, sections 2 and 3.3):

IKsubfld := 1*ia-char

ia-char := DIGIT / ALPHA / "'" / "+" / "(" / ")" /

"," / "." / "/" / "=" / "?" / "-" / "@" /
 "%" / "!" / "'" / "_" / "<" / ">"

An example "X-Recipient-ID:" field is as follows:

X-Recipient-ID: linn@ccy.bbn.com:ptf-kmc:2

This example field indicates that IA "ptf-kmc" has issued an IK component for use on messages sent to "linn@ccy.bbn.com", and that the IA has provided the number 2 as a version indicator for that IK component.

5.2.1 Subfield Definitions

The following subsections define the subfields of "X-Sender-ID:" and "X-Recipient-ID:" fields.

5.2.1.1 Entity Identifier Subfield

An entity identifier is constructed as an IKsubfld. More restrictively, an entity identifier subfield assumes the following form:

<user>@<domain-qualified-host>

In order to support universal interoperability, it is necessary to assume a universal form for the naming information. For the case of installations which transform local host names before transmission into the broader Internet, it is strongly recommended that the host

Linn

[Page 28]

RFC 1113

Mail Privacy: Procedures

August 1989

name as presented to the Internet be employed.

5.2.1.2 Issuing Authority Subfield

An IA identifier subfield is constructed as an IKsubfld. IA identifiers must be assigned in a manner which assures uniqueness. This can be done on a centralized or hierarchic basis.

5.2.1.3 Version/Expiration Subfield

A version/expiration subfield is constructed as an IKsubfld. The

version/expiration subfield format may vary among different IAs, but must satisfy certain functional constraints. An IA's version/expiration subfields must be sufficient to distinguish among the set of IK components issued by that IA for a given identified entity. Use of a monotonically increasing number is sufficient to distinguish among the IK components provided for an entity by an IA; use of a timestamp additionally allows an expiration time or date to be prescribed for an IK component.

5.2.2 IK Cryptoperiod Issues

An IK component's cryptoperiod is dictated in part by a tradeoff between key management overhead and revocation responsiveness. It would be undesirable to delete an IK component permanently before receipt of a message encrypted using that IK component, as this would render the message permanently undecipherable. Access to an expired IK component would be needed, for example, to process mail received by a user (or system) which had been inactive for an extended period of time. In order to enable very old IK components to be deleted, a message's recipient desiring encrypted local long term storage should transform the DEK used for message text encryption via re-encryption under a locally maintained IK, rather than relying on IA maintenance of old IK components for indefinite periods.

6. User Naming

6.1 Current Approach

Unique naming of electronic mail users, as is needed in order to select corresponding keys correctly, is an important topic and one which has received significant study. Our current architecture associates IK components with user names represented in a universal form ("user@domain-qualified-host"), relying on the following properties:

1. The universal form must be specifiable by an IA as it distributes IK components and known to a UA as it processes

Linn

[Page 29]

RFC 1113

Mail Privacy: Procedures

August 1989

received IK components and IK component identifiers. If a UA or IA uses addresses in a local form which is different

from the universal form, it must be able to perform an unambiguous mapping from the universal form into the local representation.

2. The universal form, when processed by a sender UA, must have a recognizable correspondence with the form of a recipient address as specified by a user (perhaps following local transformation from an alias into a universal form).

It is difficult to ensure these properties throughout the Internet. For example, an MTS which transforms address representations between the local form used within an organization and the universal form as used for Internet mail transmission may cause property 2 to be violated.

6.2 Issues for Consideration

The use of flat (non-hierarchical) electronic mail user identifiers, which are unrelated to the hosts on which the users reside, may offer value. As directory servers become more widespread, it may become appropriate for would-be senders to search for desired recipients based on such attributes. Personal characteristics, like social security numbers, might be considered. Individually-selected

identifiers could be registered with a central authority, but a means to resolve name conflicts would be necessary.

A point of particular note is the desire to accommodate multiple names for a single individual. in order to represent and allow delegation of various roles in which that individual may act. A naming mechanism that binds user roles to keys is needed. Bindings cannot be immutable since roles sometimes change (e.g., the comptroller of a corporation is fired).

It may be appropriate to examine the prospect of extending the DARPA/DoD domain system and its associated name servers to resolve user names to unique user IDs. An additional issue arises with regard to mailing list support: name servers do not currently perform (potentially recursive) expansion of lists into users. ISO and CSNet are working on user-level directory service mechanisms, which may also bear consideration.

7. Example User Interface and Implementation

In order to place the mechanisms and approaches discussed in this RFC into context, this section presents an overview of a prototype implementation. This implementation is a standalone program which is

Linn

[Page 30]

RFC 1113

Mail Privacy: Procedures

August 1989

invoked by a user, and lies above the existing UA sublayer. In the UNIX system, and possibly in other environments as well, such a program can be invoked as a "filter" within an electronic mail UA or a text editor, simplifying the sequence of operations which must be performed by the user. This form of integration offers the advantage that the program can be used in conjunction with a range of UA programs, rather than being compatible only with a particular UA.

When a user wishes to apply privacy enhancements to an outgoing message, the user prepares the message's text and invokes the standalone program (interacting with the program in order to provide address information and other data required to perform privacy enhancement processing), which in turn generates output suitable for transmission via the UA. When a user receives a privacy-enhanced message, the UA delivers the message in encrypted form, suitable for decryption and associated processing by the standalone program.

In this prototype implementation (based on symmetric key management), a cache of IK components is maintained in a local file, with entries managed manually based on information provided by originators and recipients. This cache is, effectively, a simple database. IK components are selected for transmitted messages based on the sender's identity and on recipient names, and corresponding "X-Sender-ID:" and "X-Recipient-ID:" fields are placed into the message's encapsulated header. When a message is received, these fields are used as a basis for a lookup in the database, yielding the appropriate IK component entries. DEKs and IVs are generated dynamically within the program.

Options and destination addresses are selected by command line arguments to the standalone program. The function of specifying destination addresses to the privacy enhancement program is logically distinct from the function of specifying the corresponding addresses to the UA for use by the MTS. This separation results from the fact that, in many cases, the local form of an address as specified to a UA differs from the Internet global form as used in "X-Sender-ID:" and "X-Recipient-ID:" fields.

8. Areas For Further Study

The procedures defined in this RFC are sufficient to support implementation of privacy-enhanced electronic mail transmission among cooperating parties in the Internet. Further effort will be needed, however, to enhance robustness, generality, and interoperability. In particular, further work is needed in the following areas:

1. User naming techniques, and their relationship to the domain system, name servers, directory services, and key management

Linn

[Page 31]

RFC 1113

Mail Privacy: Procedures

August 1989

functions.

2. Detailed standardization of Issuing Authority and directory service functions and interactions.
3. Privacy-enhanced interoperability with X.400 mail.

We anticipate generation of subsequent RFCs which will address these topics.

9. References

This section identifies background references which may be useful to those contemplating use of the mechanisms defined in this RFC.

ISO 7498/Part 2 - Security Architecture, prepared by ISO/TC97/SC 21/WG 1 Ad hoc group on Security, extends the OSI Basic Reference Model to cover security aspects which are general architectural elements of communications protocols, and provides an annex with tutorial and background information.

US Federal Information Processing Standards Publication (FIPS PUB) 46, Data Encryption Standard, 15 January 1977, defines the encipherment algorithm used for message text encryption and Message Authentication Code (MAC) computation.

FIPS PUB 81, DES Modes of Operation, 2 December 1980, defines specific modes in which the Data Encryption Standard algorithm may to be used to perform encryption.

FIPS PUB 113, Computer Data Authentication, May 1985, defines a specific procedure for use of the Data Encryption Standard algorithm to compute a MAC.

NOTES:

- [1] Key generation for MIC computation and message text encryption may either be performed by the sending host or by a centralized server. This RFC does not constrain this design alternative. Section 5.1 identifies possible advantages of a centralized server approach if symmetric key management is employed.
- [2] American National Standard Data Encryption Algorithm (ANSI X3.92-1981), American National Standards Institute, Approved 30 December 1980.
- [3] Federal Information Processing Standards Publication 46, Data Encryption Standard, 15 January 1977.

- [4] Information Processing Systems: Data Encipherment: Modes of Operation of a 64-bit Block Cipher.

- [5] Federal Information Processing Standards Publication 81, DES Modes of Operation, 2 December 1980.

- [6] ANSI X9.17-1985, American National Standard, Financial Institution Key Management (Wholesale), American Bankers Association, April 4, 1985, Section 7.2.

- [7] Postel, J., "Simple Mail Transfer Protocol" RFC-821, USC/Information Sciences Institute, August 1982.

- [8] This transformation should occur only at an SMTP endpoint, not at an intervening relay, but may take place at a gateway system linking the SMTP realm with other environments.

- [9] Use of the SMTP canonicalization procedure at this stage was selected since it is widely used and implemented in the Internet community, not because SMTP interoperability with this intermediate result is required; no privacy-enhanced message will

be passed to SMTP for transmission directly from this step in the four-phase transformation procedure.

- [10] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", RFC-822, August 1982.

- [11] Rose, M. and E. Stefferud, "Proposed Standard for Message Encapsulation", RFC-934, January 1985.

- [12] CCITT Recommendation X.411 (1988), "Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures".

- [13] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework".

- [14] Kille, S., "Mapping between X.400 and RFC-822", RFC-987, June 1986.

- [15] Federal Information Processing Standards Publication 113, Computer Data Authentication, May 1985.

- [16] American National Standard for Information Systems - Data

Encryption Algorithm - Modes of Operation (ANSI X3.106-1983),
American National Standards Institute - Approved 16 May 1983.

[17] Voydock, V. and S. Kent, "Security Mechanisms in High-Level

Linn

[Page 33]

RFC 1113

Mail Privacy: Procedures

August 1989

Network Protocols", ACM Computing Surveys, Vol. 15, No. 2, Pages
135-171, June 1983.

Author's Address

John Linn
Secure Systems
Digital Equipment Corporation
85 Swanson Road, BXB1-2/D04
Boxborough, MA 01719-1326

Phone: 508-264-5491

E-Mail: Linn@ultra.enet.dec.com

Linn

[Page 34]

Appendix B

RFC1114 Privacy Enhancement for Internet Electronic Mail: Part II – Certificate-Based Key Management

Network Working Group

S. Kent

Request for Comments: 1114

BBNCC

J. Linn

DEC

IAB Privacy Task Force

August 1989

Privacy Enhancement for Internet Electronic Mail:

Part II -- Certificate-Based Key Management

STATUS OF THIS MEMO

This RFC suggests a draft standard elective protocol for the Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

ACKNOWLEDGMENT

This RFC is the outgrowth of a series of IAB Privacy Task Force meetings and of internal working papers distributed for those meetings. We would like to thank the members of the Privacy Task Force for their comments and contributions at the meetings which led to the preparation of this RFC: David Balenson, Curt Barker, Matt Bishop, Morrie Gasser, Russ Housley, Dan Nessel, Mike Padlipsky, Rob Shirey, and Steve Wilbur.

Table of Contents

1. Executive Summary	2
2. Overview of Approach	3
3. Architecture	4
3.1 Scope and Restrictions	4
3.2 Relation to X.509 Architecture	7
3.3 Entities' Roles and Responsibilities	7

3.3.1 Users and User Agents	8
3.3.2 Organizational Notaries	9
3.3.3 Certification Authorities	11
3.3.3.1 Interoperation Across Certification Hierarchy Boundaries	14
3.3.3.2 Certificate Revocation	15
3.4 Certificate Definition and Usage	17
3.4.1 Contents and Use	17
3.4.1.1 Version Number	18
3.4.1.2 Serial Number	18
3.4.1.3 Subject Name	18
3.4.1.4 Issuer Name	19
3.4.1.5 Validity Period	19
3.4.1.6 Subject Public Component	20

Kent & Linn

[Page 1]

RFC 1114

Mail Privacy: Key Management

August 1989

3.4.1.7 Certificate Signature	20
3.4.2 Validation Conventions	20
3.4.3 Relation with X.509 Certificate Specification	22

1. Executive Summary

This is one of a series of RFCs defining privacy enhancement mechanisms for electronic mail transferred using Internet mail protocols. RFC-1113 (the successor to RFC 1040) prescribes protocol extensions and processing procedures for RFC-822 mail messages, given that suitable cryptographic keys are held by originators and recipients as a necessary precondition. RFC-1115 specifies algorithms for use in processing privacy-enhanced messages, as called for in RFC-1113. This RFC defines a supporting key management architecture and infrastructure, based on public-key certificate techniques, to provide keying information to message originators and recipients. A subsequent RFC, the fourth in this series, will provide detailed specifications, paper and electronic application forms, etc. for the key management infrastructure described herein.

The key management architecture described in this RFC is compatible with the authentication framework described in X.509. The major contributions of this RFC lie not in the specification of computer communication protocols or algorithms but rather in procedures and conventions for the key management infrastructure. This RFC

incorporates numerous conventions to facilitate near term implementation. Some of these conventions may be superceded in time as the motivations for them no longer apply, e.g., when X.500 or similar directory servers become well established.

The RSA cryptographic algorithm, covered in the U.S. by patents administered through RSA Data Security, Inc. (hereafter abbreviated RSADSI) has been selected for use in this key management system. This algorithm has been selected because it provides all the necessary algorithmic facilities, is "time tested" and is relatively efficient to implement in either software or hardware. It is also the primary algorithm identified (at this time) for use in international standards where an asymmetric encryption algorithm is required. Protocol facilities (e.g., algorithm identifiers) exist to permit use of other asymmetric algorithms if, in the future, it becomes appropriate to employ a different algorithm for key management. However, the infrastructure described herein is specific to use of the RSA algorithm in many respects and thus might be different if the underlying algorithm were to change.

Current plans call for RSADSI to act in concert with subscriber organizations as a "certifying authority" in a fashion described

Kent & Linn

[Page 2]

RFC 1114

Mail Privacy: Key Management

August 1989

later in this RFC. RSADSI will offer a service in which it will sign a certificate which has been generated by a user and vouched for either by an organization or by a Notary Public. This service will carry a \$25 biennial fee which includes an associated license to use the RSA algorithm in conjunction with privacy protection of electronic mail. Users who do not come under the purview of the RSA patent, e.g., users affiliated with the U.S. government or users outside of the U.S., may make use of different certifying authorities and will not require a license from RSADSI. Procedures for interacting with these other certification authorities, maintenance and distribution of revoked certificate lists from such authorities, etc. are outside the scope of this RFC. However, techniques for validating certificates issued by other authorities are contained within the RFC to ensure interoperability across the resulting jurisdictional boundaries.

2. Overview of Approach

This RFC defines a key management architecture based on the use of public-key certificates, in support of the message encipherment and authentication procedures defined in RFC-1113. In the proposed architecture, a "certification authority" representing an organization applies a digital signature to a collection of data consisting of a user's public component, various information that serves to identify the user, and the identity of the organization whose signature is affixed. (Throughout this RFC we have adopted the terms "private component" and "public component" to refer to the quantities which are, respectively, kept secret and made publically available in asymmetric cryptosystems. This convention is adopted to avoid possible confusion arising from use of the term "secret key" to refer to either the former quantity or to a key in a symmetric cryptosystem.) This establishes a binding between these user credentials, the user's public component and the organization which vouches for this binding. The resulting signed, data item is called a certificate. The organization identified as the certifying authority for the certificate is the "issuer" of that certificate.

In signing the certificate, the certification authority vouches for the user's identification, especially as it relates to the user's affiliation with the organization. The digital signature is affixed

on behalf of that organization and is in a form which can be recognized by all members of the privacy-enhanced electronic mail community. Once generated, certificates can be stored in directory servers, transmitted via unsecure message exchanges, or distributed via any other means that make certificates easily accessible to message originators, without regard for the security of the transmission medium.

Kent & Linn

[Page 3]

RFC 1114

Mail Privacy: Key Management

August 1989

Prior to sending an encrypted message, an originator must acquire a certificate for each recipient and must validate these certificates. Briefly, validation is performed by checking the digital signature in the certificate, using the public component of the issuer whose private component was used to sign the certificate. The issuer's public component is made available via some out of band means (described later) or is itself distributed in a certificate to which this validation procedure is applied recursively.

Once a certificate for a recipient is validated, the public component contained in the certificate is extracted and used to encrypt the data encryption key (DEK) that is used to encrypt the message itself.

The resulting encrypted DEK is incorporated into the X-Key-Info field of the message header. Upon receipt of an encrypted message, a recipient employs his secret component to decrypt this field, extracting the DEK, and then uses this DEK to decrypt the message.

In order to provide message integrity and data origin authentication, the originator generates a message integrity code (MIC), signs (encrypts) the MIC using the secret component of his public-key pair, and includes the resulting value in the message header in the X-MIC-Info field. The certificate of the originator is also included in the header in the X-Certificate field as described in RFC-1113, in order to facilitate validation in the absence of ubiquitous directory services. Upon receipt of a privacy enhanced message, a recipient validates the originator's certificate, extracts the public component from the certificate, and uses that value to recover (decrypt) the MIC. The recovered MIC is compared against the locally calculated MIC to verify the integrity and data origin authenticity of the message.

3. Architecture

3.1 Scope and Restrictions

The architecture described below is intended to provide a basis for managing public-key cryptosystem values in support of privacy enhanced electronic mail (see RFC-1113) in the Internet environment. The architecture describes procedures for ordering certificates from issuers, for generating and distributing certificates, and for "hot listing" of revoked certificates. Concurrent with the issuance of this RFC, RFC 1040 has been updated and reissued as RFC-1113 to describe the syntax and semantics of new or revised header fields used to transfer certificates, represent the DEK and MIC in this public-key context, and to segregate algorithm definitions into a separate RFC to facilitate the addition of other algorithms in the future. This RFC focuses on the management aspects of certificate-

based, public-key cryptography for privacy enhanced mail while RFC-1113 addresses representation and processing aspects of such mail, including changes required by this key management technology.

The proposed architecture imposes conventions for certification paths which are not strictly required by the X.509 recommendation nor by the technology itself. The decision to impose these conventions is based in part on constraints imposed by the status of the RSA cryptosystem within the U.S. as a patented algorithm, and in part on the need for an organization to assume operational responsibility for certificate management in the current (minimal) directory system infrastructure for electronic mail. Over time, we anticipate that some of these constraints, e.g., directory service availability, will change and the procedures specified in the RFC will be reviewed and modified as appropriate.

At this time, we propose a system in which user certificates represent the leaves in a shallow (usually two tier) certification hierarchy (tree). Organizations which act as issuers are represented by certificates higher in the tree. This convention minimizes the complexity of validating user certificates by limiting the length of "certification paths" and by making very explicit the relationship

between a certificate issuer and a user. Note that only organizations may act as issuers in the proposed architecture; a user certificate may not appear in a certification path, except as the terminal node in the path. These conventions result in a certification hierarchy which is a compatible subset of that permitted under X.509, with respect to both syntax and semantics.

The RFC proposes that RSADSI act as a "co-issuer" of certificates on behalf of most organizations. This can be effected in a fashion which is "transparent" so that the organizations appear to be the issuers with regard to certificate formats and validation procedures. This is effected by having RSADSI generate and hold the secret components used to sign certificates on behalf of organizations. The motivation for RSADSI's role in certificate signing is twofold. First, it simplifies accounting controls in support of licensing, ensuring that RSADSI is paid for each certificate. Second, it contributes to the overall integrity of the system by establishing a uniform, high level of protection for the private-components used to sign certificates. If an organization were to sign certificates directly on behalf of its affiliated users, the organization would have to establish very stringent security and accounting mechanisms and enter into (elaborate) legal agreements with RSADSI in order to provide a comparable level of assurance. Requests by organizations

to perform direct certificate signing will be considered on a case-by-case basis, but organizations are strongly urged to make use of the facilities proposed by this RFC.

Kent & Linn

[Page 5]

RFC 1114

Mail Privacy: Key Management

August 1989

Note that the risks associated with disclosure of an organization's secret component are different from those associated with disclosure of a user's secret component. The former component is used only to sign certificates, never to encrypt message traffic. Thus the exposure of an organization's secret component could result in the generation of forged certificates for users affiliated with that organization, but it would not affect privacy-enhanced messages which are protected using legitimate certificates. Also note that any certificates generated as a result of such a disclosure are readily traceable to the issuing authority which holds this component, e.g., RSADSI, due to the non-repudiation feature of the digital signature. The certificate registration and signing procedures established in this RFC would provide non-repudiable evidence of disclosure of an

organization's secret component by RSADSI. Thus this RFC advocates use of RSADSI as a co-issuer for certificates until such time as technical security mechanisms are available to provide a similar, system-wide level of assurance for (distributed) certificate signing by organizations.

We identify two classes of exceptions to this certificate signing paradigm. First, the RSA algorithm is patented only within the U.S., and thus it is very likely that certificate signing by issuers will arise outside of the U.S., independent of RSADSI. Second, the research that led to the RSA algorithm was sponsored by the National Science Foundation, and thus the U.S. government retains royalty-free license rights to the algorithm. Thus the U.S. government may establish a certificate generation facilities for its affiliated users. A number of the procedures described in this document apply only to the use of RSADSI as a certificate co-issuer; all other certificate generation practices lie outside the scope of this RFC.

This RFC specifies procedures by which users order certificates either directly from RSADSI or via a representative in an organization with which the user holds some affiliation (e.g., the user's employer or educational institution). Syntactic provisions are made which allow a recipient to determine, to some granularity,

which identifying information contained in the certificate is vouched for by the certificate issuer. In particular, organizations will usually be vouching for the affiliation of a user with that organization and perhaps a user's role within the organization, in addition to the user's name. In other circumstances, as discussed in section 3.3.3, a certificate may indicate that an issuer vouches only for the user's name, implying that any other identifying information contained in the certificate may not have been validated by the issuer. These semantics are beyond the scope of X.509, but are not incompatible with that recommendation.

The key management architecture described in this RFC has been

Kent & Linn

[Page 6]

RFC 1114

Mail Privacy: Key Management

August 1989

designed to support privacy enhanced mail as defined in this RFC, RFC-1113, and their successors. Note that this infrastructure also supports X.400 mail security facilities (as per X.411) and thus paves the way for transition to the OSI/CCITT Message Handling System

paradigm in the Internet in the future. The certificate issued to a user for the \$25 biennial fee will grant to the user identified by that certificate a license from RSADSI to employ the RSA algorithm for certificate validation and for encryption and decryption operations in this electronic mail context. No use of the algorithm outside the scope defined in this RFC is authorized by this license as of this time. Expansion of the license to other Internet security applications is possible but not yet authorized. The license granted by this fee does not authorize the sale of software or hardware incorporating the RSA algorithm; it is an end-user license, not a developer's license.

3.2 Relation to X.509 Architecture

CCITT 1988 Recommendation X.509, "The Directory - Authentication Framework", defines a framework for authentication of entities involved in a distributed directory service. Strong authentication, as defined in X.509, is accomplished with the use of public-key cryptosystems. Unforgeable certificates are generated by certification authorities; these authorities may be organized hierarchically, though such organization is not required by X.509. There is no implied mapping between a certification hierarchy and the naming hierarchy imposed by directory system naming attributes. The

public-key certificate approach defined in X.509 has also been adopted in CCITT 1988 X.411 in support of the message handling application.

This RFC interprets the X.509 certificate mechanism to serve the needs of privacy-enhanced mail in the Internet environment. The certification hierarchy proposed in this RFC in support of privacy enhanced mail is intentionally a subset of that allowed under X.509. In large part constraints have been levied in order to simplify certificate validation in the absence of a widely available, user-level directory service. The certification hierarchy proposed here also embodies semantics which are not explicitly addressed by X.509, but which are consistent with X.509 precepts. The additional semantic constraints have been adopted to explicitly address questions of issuer "authority" which we feel are not well defined in X.509.

3.3 Entities' Roles and Responsibilities

One way to explain the architecture proposed by this RFC is to examine the various roles which are defined for various entities in

Kent & Linn

[Page 7]

RFC 1114

Mail Privacy: Key Management

August 1989

the architecture and to describe what is required of each entity in order for the proposed system to work properly. The following sections identify three different types of entities within this architecture: users and user agents, organizational notaries, and certification authorities. For each class of entity we describe the (electronic and paper) procedures which the entity must execute as part of the architecture and what responsibilities the entity assumes as a function of its role in the architecture. Note that the infrastructure described here applies to the situation wherein RSADSI acts as a co-issuer of certificates, sharing the role of certification authority as described later. Other certifying authority arrangements may employ different procedures and are not addressed by this RFC.

3.3.1 Users and User Agents

The term User Agent (UA) is taken from CCITT X.400 Message Handling Systems (MHS) Recommendations, which define it as follows: "In the

context of message handling, the functional object, a component of MHS, by means of which a single direct user engages in message handling." UAs exchange messages by calling on a supporting Message Transfer Service (MTS).

A UA process supporting privacy-enhanced mail processing must protect the private component of its associated entity (ordinarily, a human user) from disclosure. We anticipate that a user will employ ancillary software (not otherwise associated with the UA) to generate his public/private component pair and to compute the (one-way) message hash required by the registration procedure. The public component, along with information that identifies the user, will be transferred to an organizational notary (see below) for inclusion in an order to an issuer. The process of generating public and private components is a local matter, but we anticipate Internet-wide distribution of software suitable for component-pair generation to facilitate the process. The mechanisms used to transfer the public component and the user identification information must preserve the integrity of both quantities and bind the two during this transfer.

This proposal establishes two ways in which a user may order a certificate, i.e., through the user's affiliation with an organization or directly through RSADSI. In either case, a user will

be required to send a paper order to RSADSI on a form described in a subsequent RFC and containing the following information:

1. Distinguished Name elements (e.g., full legal name, organization name, etc.)
2. Postal address

Kent & Linn

[Page 8]

RFC 1114

Mail Privacy: Key Management

August 1989

3. Internet electronic mail address
4. A message hash function, binding the above information to the user's public component

Note that the user's public component is NOT transmitted via this paper path. In part the rationale here is that the public component consists of many (>100) digits and thus is prone to error if it is copied to and from a piece of paper. Instead, a message hash is

computed on the identifying information and the public component and this (smaller) message hash value is transmitted along with the identifying information. Thus the public component is transferred only via an electronic path, as described below.

If the user is not affiliated with an organization which has established its own "electronic notary" capability (an organization notary or "ON" as discussed in the next section), then this paper registration form must be notarized by a Notary Public. If the user is affiliated with an organization which has established one or more ONs, the paper registration form need not carry the endorsement of a Notary Public. Concurrent with the paper registration, the user must send the information outlined above, plus his public component, either to his ON, or directly to RSADSI if no appropriate ON is available to the user. Direct transmission to RSADSI of this information will be via electronic mail, using a representation described in a subsequent RFC. The paper registration must be accompanied by a check or money order for \$25 or an organization may establish some other billing arrangement with RSADSI. The maximum (and default) lifetime of a certificate ordered through this process is two years.

The transmission of ID information and public component from a user

to his ON is a local matter, but we expect electronic mail will also be the preferred approach in many circumstances and we anticipate general distribution of software to support this process. Note that it is the responsibility of the user and his organization to ensure the integrity of this transfer by some means deemed adequately secure for the local computing and communication environment. There is no requirement for secrecy in conjunction with this information transfer, but the integrity of the information must be ensured.

3.3.2 Organizational Notaries

An organizational notary is an individual who acts as a clearinghouse for certificate orders originating within an administrative domain such as a corporation or a university. An ON represents an organization or organizational unit (in X.500 naming terms), and is assumed to have some independence from the users on whose behalf

certificates are ordered. An ON will be restricted through mechanisms implemented by the issuing authority, e.g., RSADSI, to ordering certificates properly associated with the domain of that ON. For example, an ON for BBN should not be able to order certificates for users affiliated with MIT or MITRE, nor vice versa. Similarly, if a corporation such as BBN were to establish ONs on a per-subsidary basis (corresponding to organization units in X.500 naming parlance), then an ON for the BBN Communications subsidiary should not be allowed to order a certificate for a user who claims affiliation with the BBN Software Products subsidiary.

It can be assumed that the set of ONs changes relatively slowly and that the number of ONs is relatively small in comparison with the number of users. Thus a more extensive, higher assurance process may reasonably be associated with ON accreditation than with per-user certificate ordering. Restrictions on the range of information which an ON is authorized to certify are established as part of this more elaborate registration process. The procedures by which organizations and organizational units are established in the RSADSI database, and by which ONs are registered, will be described in a subsequent RFC.

An ON is responsible for establishing the correctness and integrity of information incorporated in an order, and will generally vouch for (certify) the accuracy of identity information at a granularity finer than that provided by a Notary Public. We do not believe that it is feasible to enforce uniform standards for the user certification process across all ONs, but we anticipate that organizations will endeavor to maintain high standards in this process in recognition of the "visibility" associated with the identification data contained in certificates. An ON also may constrain the validity period of an ordered certificate, restricting it to less than the default two year interval imposed by the RSADSI license agreement.

An ON participates in the certificate ordering process by accepting and validating identification information from a user and forwarding this information to RSADSI. The ON accepts the electronic ordering information described above (Distinguished Name elements, mailing address, public component, and message hash computed on all of this data) from a user. (The representation for user-to-ON transmission of this data is a local matter, but we anticipate that the encoding specified for ON-to-RSADSI representation of this data will often be employed.) The ON sends an integrity-protected (as described in RFC-1113) electronic message to RSADSI, vouching for the correctness of the binding between the public component and the identification

data. Thus, to support this function, each ON will hold a certificate as an individual user within the organization which he represents. RSADSI will maintain a database which identifies the

Kent & Linn

[Page 10]

RFC 1114

Mail Privacy: Key Management

August 1989

users who also act as ONs and the database will specify constraints on credentials which each ON is authorized to certify. The electronic mail representation for a user's certificate data in an ON message to RSADSI will be specified in a subsequent RFC.

3.3.3 Certification Authorities

In X.509 the term "certification authority" is defined as "an authority trusted by one or more users to create and assign certificates". This alternate expansion for the acronym "CA" is roughly equivalent to that contemplated as a "central authority" in RFC-1040 and RFC-1113. The only difference is that in X.509 there is no requirement that a CA be a distinguished entity or that a CA serve

a large number of users, as envisioned in these RFCs. Rather, any user who holds a certificate can, in the X.509 context, act as a CA for any other user. As noted above, we have chosen to restrict the role of CA in this electronic mail environment to organizational entities, to simplify the certificate validation process, to impose semantics which support organizational affiliation as a basis for certification, and to facilitate license accountability.

In the proposed architecture, individuals who are affiliated with (registered) organizations will go through the process described above, in which they forward their certificate information to their ON for certification. The ON will, based on local procedures, verify the accuracy of the user's credentials and forward this information to RSADSI using privacy-enhanced mail to ensure the integrity and authenticity of the information. RSADSI will carry out the actual certificate generation process on behalf of the organization represented by the ON. Recall that it is the identity of the organization which the ON represents, not the ON's identity, which appears in the issuer field of the user certificate. Therefore it is the private component of the organization, not the ON, which is used to sign the user certificate.

In order to carry out this procedure RSADSI will serve as the

repository for the private components associated with certificates representing organizations or organizational units (but not individuals). In effect the role of CA will be shared between the organizational notaries and RSADSI. This shared role will not be visible in the syntax of the certificates issued under this arrangement nor is it apparent from the validation procedure one applies to these certificates. In this sense, the role of RSADSI as the actual signer of certificates on behalf of organizations is transparent to this aspect of system operation.

If an organization were to carry out the certificate signing process locally, and thus hold the private component associated with its

Kent & Linn

[Page 11]

RFC 1114

Mail Privacy: Key Management

August 1989

organization certificate, it would need to contact RSADSI to discuss security safeguards, special legal agreements, etc. A number of requirements would be imposed on an organization if such an approach were pursued. The organization would be required to execute

additional legal instruments with RSADSI, e.g., to ensure proper accounting for certificates generated by the organization. Special software will be required to support the certificate signing process, distinct from the software required for an ON. Stringent procedural, physical, personnel and computer security safeguards would be required to support this process, to maintain a relatively high level of security for the system as a whole. Thus, at this time, it is not recommended that organizations pursue this approach although local certificate generation is not expressly precluded by the proposed architecture.

RSADSI has offered to operate a service in which it serves as a CA for users who are not affiliated with any organization or who are affiliated with an organization which has not opted to establish an organizational notary. To distinguish certificates issued to such "non-affiliated" users the distinguished string "Notary" will appear as the organizational unit name of the issuer of the certificate. This convention will be employed throughout the system. Thus not only RSADSI but any other organization which elects to provide this type of service to non-affiliated users may do so in a standard fashion. Hence a corporation might issue a certificate with the "Notary" designation to students hired for the summer, to differentiate them from full-time employees. At least in the case of

RSADSI, the standards for verifying user credentials that carry this designation will be well known and widely recognized (e.g., Notary Public endorsement).

To illustrate this convention, consider the following examples.

Employees of RSADSI will hold certificates which indicate "RSADSI" as the organization in both the issuer field and the subject field, perhaps with no organizational unit specified. Certificates obtained directly from RSADSI, by user's who are not affiliated with any ON, will also indicate "RSADSI" as the organization and will specify "Notary" as an organizational unit in the issuer field. However, these latter certificates will carry some other designation for organization (and, optionally, organizational unit) in the subject field. Moreover, an organization designated in the subject field for such a certificate will not match any for which RSADSI has an ON registered (to avoid possible confusion).

In all cases described above, when a certificate is generated RSADSI will send a paper reply to the ordering user, including two message hash functions:

Kent & Linn

[Page 12]

RFC 1114

Mail Privacy: Key Management

August 1989

1. a message hash computed on the user's identifying information and public component (and sent to RSADSI in the registration process), to guarantee its integrity across the ordering process, and
2. a message hash computed on the public component of RSADSI, to provide independent authentication for this public component which is transmitted to the user via email (see below).

RSADSI will send to the user via electronic mail (not privacy enhanced) a copy of his certificate, a copy of the organization certificate identified in the issuer field of the user's certificate, and the public component used to validate certificates signed by RSADSI. The "issuer" certificate is included to simplify the validation process in the absence of a user-level directory system; its distribution via this procedure will probably be phased out in the future. Thus, as described in RFC-1113, the originator of a message is encouraged, though not required, to include his

certificate, and that of its issuer, in the privacy enhanced message header (X-Issuer-Certificate) to ensure that each recipient can process the message using only the information contained in this header. The organization (organizational unit) identified in the subject field of the issuer certificate should correspond to that which the user claims affiliation (as declared in the subject field of his certificate). If there is no appropriate correspondence between these fields, recipients ought to be suspicious of the implied certification path. This relationship should hold except in the case of "non-affiliated" users for whom the "Notary" convention is employed.

In contrast, the issuer field of the issuer's certificate will specify "RSADSI" as the organization, i.e., RSADSI will certify all organizational certificates. This convention allows a recipient to validate any originator's certificate (within the RSADSI certification hierarchy) in just two steps. Even if an organization establishes a certification hierarchy involving organizational units, certificates corresponding to each unit can be certified both by RSADSI and by the organizational entity immediately superior to the unit in the hierarchy, so as to preserve this short certification path feature. First, the public component of RSADSI is employed to validate the issuer's certificate. Then the issuer's public

component is extracted from that certificate and is used to validate the originator's certificate. The recipient then extracts the originator's public component for use in processing the X-Mic-Info field of the message (see and RFC-1113).

The electronic representation used for transmission of the data items described above (between an ON and RSADSI) will be contained in a

Kent & Linn

[Page 13]

RFC 1114

Mail Privacy: Key Management

August 1989

subsequent RFC. To verify that the registration process has been successfully completed and to prepare for exchange of privacy-enhanced electronic mail, the user should perform the following steps:

1. extract the RSADSI public component, the issuer's certificate and the user's certificate from the message
2. compute the message hash on the RSADSI public component and

compare the result to the corresponding message hash that was included in the paper receipt

3. use the RSADSI public component to validate the signature on the issuer's certificate (RSADSI will be the issuer of this certificate)
4. extract the organization public component from the validated issuer's certificate and use this public component to validate the user certificate
5. extract the identification information and public component from the user's certificate, compute the message hash on it and compare the result to the corresponding message hash value transmitted via the paper receipt

For a user whose order was processed via an ON, successful completion of these steps demonstrates that the certificate issued to him matches that which he requested and which was certified by his ON. It also demonstrates that he possesses the (correct) public component for RSADSI and for the issuer of his certificate. For a user whose order was placed directly with RSADSI, this process demonstrates that his certificate order was properly processed by RSADSI and that he

possesses the valid issuer certificate for the RSADSI Notary. The user can use the RSADSI public component to validate organizational certificates for organizations other than his own. He can employ the public component associated with his own organization to validate certificates issued to other users in his organization.

3.3.3.1 Interoperation Across Certification Hierarchy Boundaries

In order to accommodate interoperation with other certification authorities, e.g., foreign or U.S. government CAs, two conventions will be adopted. First, all certifying authorities must agree to "cross-certify" one another, i.e., each must be willing to sign a certificate in which the issuer is that certifying authority and the subject is another certifying authority. Thus, RSADSI might generate a certificate in which it is identified as the issuer and a certifying authority for the U.S. government is identified as the

subject. Conversely, that U.S. government certifying authority would generate a certificate in which it is the issuer and RSADSI is the subject. This cross-certification of certificates for "top-level" CAs establishes a basis for "lower level" (e.g., organization and user) certificate validation across the hierarchy boundaries. This avoids the need for users in one certification hierarchy to engage in some "out-of-band" procedure to acquire a public-key for use in validating certificates from a different certification hierarchy.

The second convention is that more than one X-Issuer-Certificate field may appear in a privacy-enhanced mail header. Multiple issuer certificates can be included so that a recipient can more easily validate an originator's certificate when originator and recipient are not part of a common CA hierarchy. Thus, for example, if an originator served by the RSADSI certification hierarchy sends a message to a recipient served by a U.S. government hierarchy, the originator could (optionally) include an X-Issuer-Certificate field containing a certificate issued by the U.S. government CA for RSADSI. In this fashion the recipient could employ his public component for the U.S. government CA to validate this certificate for RSADSI, from which he would extract the RSADSI public component to validate the certificate for the originator's organization, from which he would extract the public component required to validate the originator's

certificate. Thus, more steps can be required to validate certificates when certification hierarchy boundaries are crossed, but the same basic procedure is employed. Remember that caching of certificates by UAs can significantly reduce the effort required to process messages and so these examples should be viewed as "worse case" scenarios.

3.3.3.2 Certificate Revocation

X.509 states that it is a CA's responsibility to maintain:

1. a time-stamped list of the certificates it issued which have been revoked
2. a time-stamped list of revoked certificates representing other CAs

There are two primary reasons for a CA to revoke a certificate, i.e., suspected compromise of a secret component (invalidating the corresponding public component) or change of user affiliation (invalidating the Distinguished Name). As described in X.509, "hot listing" is one means of propagating information relative to certificate revocation, though it is not a perfect mechanism. In

particular, an X.509 Revoked Certificate List (RCL) indicates only the age of the information contained in it; it does not provide any

Kent & Linn

[Page 15]

RFC 1114

Mail Privacy: Key Management

August 1989

basis for determining if the list is the most current RCL available from a given CA. To help address this concern, the proposed architecture establishes a format for an RCL in which not only the date of issue, but also the next scheduled date of issue is specified. This is a deviation from the format specified in X.509.

Adopting this convention, when the next scheduled issue date arrives a CA must issue a new RCL, even if there are no changes in the list of entries. In this fashion each CA can independently establish and advertise the frequency with which RCLs are issued by that CA. Note that this does not preclude RCL issuance on a more frequent basis, e.g., in case of some emergency, but no Internet-wide mechanisms are architected for alerting users that such an unscheduled issuance has taken place. This scheduled RCL issuance convention allows users

(UAs) to determine whether a given RCL is "out of date," a facility not available from the standard RCL format.

A recent (draft) version of the X.509 recommendation calls for each RCL to contain the serial numbers of certificates which have been revoked by the CA administering that list, i.e., the CA that is identified as the issuer for the corresponding revoked certificates. Upon receipt of a RCL, a UA should compare the entries against any cached certificate information, deleting cache entries which match RCL entries. (Recall that the certificate serial numbers are unique only for each issuer, so care must be exercised in effecting this cache search.) The UA should also retain the RCL to screen incoming messages to detect use of revoked certificates carried in these message headers. More specific details for processing RCL are beyond the scope of this RFC as they are a function of local certificate management techniques.

In the architecture defined by this RFC, a RCL will be maintained for each CA (organization or organizational unit), signed using the private component of that organization (and thus verifiable using the public component of that organization as extracted from its certificate). The RSADSI Notary organizational unit is included in this collection of RCLs. CAs operated under the auspices of the U.S.

government or foreign CAs are requested to provide RCLs conforming to these conventions, at least until such time as X.509 RCLs provide equivalent functionality, in support of interoperability with the Internet community. An additional, "top level" RCL, will be maintained by RSAD-SI, and should be maintained by other "top level" CAs, for revoked organizational certificates.

The hot listing procedure (expect for this top level RCL) will be effected by having an ON from each organization transmit to RSADSI a list of the serial numbers of users within his organization, to be hot listed. This list will be transmitted using privacy-enhanced

Kent & Linn

[Page 16]

RFC 1114

Mail Privacy: Key Management

August 1989

mail to ensure authenticity and integrity and will employ representation conventions to be provided in a subsequent RFC. RSADSI will format the RCL, sign it using the private component of the organization, and transmit it to the ON for dissemination, using a representation defined in a subsequent RFC. Means for

dissemination of RCLs, both within the administrative domain of a CA and across domain boundaries, are not specified by this proposal. However, it is anticipated that each hot list will also be available via network information center databases, directory servers, etc.

The following ASN.1 syntax, derived from X.509, defines the format of RCLs for use in the Internet privacy enhanced email environment. See the ASN.1 definition of certificates (later in this RFC or in X.509, Annex G) for comparison.

```
revokedCertificateList ::= SIGNED SEQUENCE {  
    signature      AlgorithmIdentifier,  
    issuer         Name,  
    list           SEQUENCE RCLEntry,  
    lastUpdate     UTCTime,  
    nextUpdate     UTCTime}
```

```
RCLEntry ::= SEQUENCE {  
    subject         CertificateSerialNumber,  
    revocationDate  UTCTime}
```

3.4 Certificate Definition and Usage

3.4.1 Contents and Use

A certificate contains the following contents:

1. version
2. serial number
3. certificate signature (and associated algorithm identifier)
4. issuer name
5. validity period
6. subject name
7. subject public component (and associated algorithm identifier)

This section discusses the interpretation and use of each of these certificate elements.

3.4.1.1 Version Number

The version number field is intended to facilitate orderly changes in certificate formats over time. The initial version number for certificates is zero (0).

3.4.1.2 Serial Number

The serial number field provides a short form, unique identifier for each certificate generated by an issuer. The serial number is used in RCLs to identify revoked certificates instead of including entire certificates. Thus each certificate generated by an issuer must contain a unique serial number. It is suggested that these numbers be issued as a compact, monotonic increasing sequence.

3.4.1.3 Subject Name

A certificate provides a representation of its subject's identity and organizational affiliation in the form of a Distinguished Name. The

fundamental binding ensured by the privacy enhancement mechanisms is that between public-key and the user identity. CCITT Recommendation X.500 defines the concept of Distinguished Name.

Version 2 of the U.S. Government Open Systems Interconnection Profile (GOSIP) specifies maximum sizes for O/R Name attributes. Since most of these attributes also appear in Distinguished Names, we have adopted the O/R Name attribute size constraints specified in GOSIP and noted below. Using these size constraints yields a maximum Distinguished Name length (exclusive of ASN encoding) of two-hundred fifty-nine (259) characters, based on the required and optional attributes described below for subject names. The following attributes are required in subject Distinguished Names for purposes of this RFC:

1. Country Name in standard encoding (e.g., the two-character Printable String "US" assigned by ISO 3166 as the identifier for the United States of America, the string "GB" assigned as the identifier for the United Kingdom, or the string "NQ" assigned as the identifier for Dronning Maud Land). Maximum ASCII character length of three (3).
2. Organizational Name (e.g., the Printable String "Bolt Beranek

and Newman, Inc."). Maximum ASCII character length of sixty-four (64).

3. Personal Name (e.g., the X.402/X.411 structured Printable String encoding for the name John Linn). Maximum ASCII character length of sixty-four (64).

Kent & Linn

[Page 18]

RFC 1114

Mail Privacy: Key Management

August 1989

The following attributes are optional in subject Distinguished Names for purposes of this RFC:

1. Organizational Unit Name(s) (e.g., the Printable String "BBN Communications Corporation") A hierarchy of up to four organizational unit names may be provided; the least significant member of the hierarchy is represented first. Each of these attributes has a maximum ASCII character length of thirty-two (32), for a total of one-hundred and twenty-eight

(128) characters if all four are present.

3.4.1.4 Issuer Name

A certificate provides a representation of its issuer's identity, in the form of a Distinguished Name. The issuer identification is needed in order to determine the appropriate issuer public component to use in performing certificate validation. The following attributes are required in issuer Distinguished Names for purposes of this RFC:

1. Country Name (e.g., encoding for "US")
2. Organizational Name

The following attributes are optional in issuer Distinguished Names for purposes of this RFC:

1. Organizational Unit Name(s). (A hierarchy of up to four organizational unit names may be provided; the least significant member of the hierarchy is represented first.) If the issuer is vouching for the user identity in the Notary capacity described above, then exactly one instance of this field

must be present and it must consist of the string "Notary".

As noted earlier, only organizations are allowed as issuers in the proposed authentication hierarchy. Hence the Distinguished Name for an issuer should always be that of an organization, not a user, and thus no Personal Name field may be included in the Distinguished Name of an issuer.

3.4.1.5 Validity Period

A certificate carries a pair of time specifiers, indicating the start and end of the time period over which a certificate is intended to be used. No message should ever be prepared for transmission with a non-current certificate, but recipients should be prepared to receive messages processed using recently-expired certificates. This fact results from the unpredictable (and sometimes substantial)

transmission delay of the staged-delivery electronic mail environment. The default and maximum validity period for certificates issued in this system will be two years.

3.4.1.6 Subject Public Component

A certificate carries the public component of its associated entity, as well as an indication of the algorithm with which the public component is to be used. For purposes of this RFC, the algorithm identifier will indicate use of the RSA algorithm, as specified in RFC-1115. Note that in this context, a user's public component is actually the modulus employed in RSA algorithm calculations. A "universal" (public) exponent is employed in conjunction with the modulus to complete the system. Two choices of exponents are recommended for use in this context and are described in section 3.4.3. Modulus size will be permitted to vary between 320 and 632 bits.

3.4.1.7 Certificate Signature

A certificate carries a signature algorithm identifier and a signature, applied to the certificate by its issuer. The signature is validated by the user of a certificate, in order to determine that

the integrity of its contents have not been compromised subsequent to generation by a CA. An encrypted, one-way hash will be employed as the signature algorithm. Hash functions suitable for use in this context are notoriously difficult to design and tend to be computationally intensive. Initially we have adopted a hash function developed by RSADSI and which exhibits performance roughly equivalent to the DES (in software). This same function has been selected for use in other contexts in this system where a hash function (message hash algorithm) is required, e.g., MIC for multicast messages. In the future we expect other one-way hash functions will be added to the list of algorithms designated for this purpose.

3.4.2 Validation Conventions

Validating a certificate involves verifying that the signature affixed to the certificate is valid, i.e., that the hash value computed on the certificate contents matches the value that results from decrypting the signature field using the public component of the issuer. In order to perform this operation the user must possess the public component of the issuer, either via some integrity-assured channel, or by extracting it from another (validated) certificate. In the proposed architecture this recursive operation is terminated quickly by adopting the convention that RSADSI will certify the

certificates of all organizations or organizational units which act as issuers for end users. (Additional validation steps may be

Kent & Linn

[Page 20]

RFC 1114

Mail Privacy: Key Management

August 1989

required for certificates issued by other CAs as described in section 3.3.3.1.)

Certification means that RSADSI will sign certificates in which the subject is the organization or organizational unit and for which RSADSI is the issuer, thus implying that RSADSI vouches for the credentials of the subject. This is an appropriate construct since each ON representing an organization or organizational unit must have registered with RSADSI via a procedure more rigorous than individual user registration. This does not preclude an organizational unit from also holding a certificate in which the "parent" organization (or organizational unit) is the issuer. Both certificates are appropriate and permitted in the X.509 framework. However, in order to facilitate the validation process in an environment where user-

level directory services are generally not available, we will (at this time) adopt this certification convention.

The public component needed to validate certificates signed by RSADSI (in its role as a CA for issuers) is transmitted to each user as part of the registration process (using electronic mail with independent, postal confirmation via a message hash). Thus a user will be able to validate any user certificate (from the RSADSI hierarchy) in at most two steps. Consider the situation in which a user receives a privacy enhanced message from an originator with whom the recipient has never previously corresponded. Based on the certification convention described above, the recipient can use the RSADSI public component to validate the issuer's certificate contained in the X-Issuer-Certificate field. (We recommend that, initially, the originator include his organization's certificate in this optional field so that the recipient need not access a server or cache for this public component.) Using the issuer's public component (extracted from this certificate), the recipient can validate the originator's certificate contained in the X-Certificate field of the header.

Having performed this certificate validation process, the recipient can extract the originator's public component and use it to decrypt the content of the X-MIC-Info field and thus verify the data origin

authenticity and integrity of the message. Of course, implementations of privacy enhanced mail should cache validated public components (acquired from incoming mail or via the message from a user registration process) to speed up this process. If a message arrives from an originator whose public component is held in the recipient's cache, the recipient can immediately employ that public component without the need for the certificate validation process described here. Also note that the arithmetic required for certificate validation is considerably faster than that involved in digitally signing a certificate, so as to minimize the computational burden on users.

Kent & Linn

[Page 21]

RFC 1114

Mail Privacy: Key Management

August 1989

A separate issue associated with validation of certificates is a semantic one, i.e., is the entity identified in the issuer field appropriate to vouch for the identifying information in the subject field. This is a topic outside the scope of X.509, but one which must be addressed in any viable system. The hierarchy proposed in

this RFC is designed to address this issue. In most cases a user will claim, as part of his identifying information, affiliation with some organization and that organization will have the means and responsibility for verifying this identifying information. In such circumstances one should expect an obvious relationship between the Distinguished Name components in the issuer and subject fields.

For example, if the subject field of a certificate identified an individual as affiliated with the "Widget Systems Division" (Organizational Unit Name) of "Compudigicorp" (Organizational Name), one would expect the issuer field to specify "Compudigicorp" as the Organizational Name and, if an Organizational Unit Name were present, it should be "Widget Systems Division." If the issuer's certificate indicated "Compudigicorp" as the subject (with no Organizational Unit specified), then the issuer should be "RSADSI." If the issuer's certificate indicated "Widget Systems Division" as Organizational Unit and "Compudigicorp" as Organization in the subject field, then the issuer could be either "RSADSI" (due to the direct certification convention described earlier) or "Compudigicorp" (if the organization elected to distribute this intermediate level certificate). In the later case, the certificate path would involve an additional step using the certificate in which "Compudigicorp" is the subject and "RSADSI" is the issuer. One should be suspicious if the validation

path does not indicate a subset relationship for the subject and issuer Distinguished Names in the certification path, expect where cross-certification is employed to cross CA boundaries.

It is a local matter whether the message system presents a human user with the certification path used to validate a certificate associated with incoming, privacy-enhanced mail. We note that a visual display of the Distinguished Names involved in that path is one means of providing the user with the necessary information. We recommend, however, that certificate validation software incorporate checks and alert the user whenever the expected certification path relationships are not present. The rationale here is that regular display of certification path data will likely be ignored by users, whereas automated checking with a warning provision is a more effective means of alerting users to possible certification path anomalies. We urge developers to provide facilities of this sort.

3.4.3 Relation with X.509 Certificate Specification

An X.509 certificate can be viewed as two components: contents and an

encrypted hash. The encrypted hash is formed and processed as follows:

1. X, the hash, is computed as a function of the certificate contents
2. the hash is signed by raising X to the power e (modulo n)
3. the hash's signature is validated by raising the result of step 2 to the power d (modulo n), yielding X, which is compared with the result computed as a function of certificate contents.

Annex C to X.509 suggests the use of Fermat number F4 (65537 decimal, $1 + 2^{2^{16}}$) as a fixed value for e which allows relatively efficient authentication processing, i.e., at most seventeen (17) multiplications are required to effect exponentiation). As an alternative one can employ three (3) as the value for e, yielding even faster exponentiation, but some precautions must be observed

(see RFC-1115). Users of the algorithm select values for d (a secret quantity) and n (a non-secret quantity) given this fixed value for e . As noted earlier, this RFC proposes that either three (3) or F4 be employed as universal encryption exponents, with the choice specified in the algorithm identifier. In particular, use of an exponent value of three (3) for certificate validation is encouraged, to permit rapid certificate validation. Given these conventions, a user's public component, and thus the quantity represented in his certificate, is actually the modulus (n) employed in this computation (and in the computations used to protect the DEK and MSGHASH, as described in RFC-1113). A user's private component is the exponent (d) cited above.

The X.509 certificate format is defined (in X.509, Annex G) by the following ASN.1 syntax:

```
Certificate ::= SIGNED SEQUENCE{
    version [0]      Version DEFAULT v1988,
    serialNumber      CertificateSerialNumber,
    signature          AlgorithmIdentifier,
    issuer             Name,
    validity           Validity,
    subject            Name,
```

subjectPublicKeyInfo SubjectPublicKeyInfo}

Version ::= INTEGER {v1988(0)}

CertificateSerialNumber ::= INTEGER

Kent & Linn

[Page 23]

RFC 1114

Mail Privacy: Key Management

August 1989

Validity ::= SEQUENCE{
 notBefore UTCTime,
 notAfter UTCTime}

SubjectPublicKeyInfo ::= SEQUENCE{
 algorithm AlgorithmIdentifier,
 subjectPublicKey BIT STRING}

AlgorithmIdentifier ::= SEQUENCE{

algorithm OBJECT IDENTIFIER,
parameters ANY DEFINED BY algorithm OPTIONAL}

All components of this structure are well defined by ASN.1 syntax defined in the 1988 X.400 and X.500 Series Recommendations, except for the AlgorithmIdentifier. An algorithm identifier for RSA is contained in Annex H of X.509 but is unofficial. RFC-1115 will provide detailed syntax and values for this field.

NOTES:

- [1] CCITT Recommendation X.411 (1988), "Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures".
- [2] CCITT Recommendation X.509 (1988), "The Directory Authentication Framework".

Authors' Addresses

Steve Kent

BBN Communications

50 Moulton Street

Cambridge, MA 02138

Phone: (617) 873-3988

E-Mail: kent@BBN.COM

John Linn

Secure Systems

Digital Equipment Corporation

85 Swanson Road, BXB1-2/D04

Boxborough, MA 01719-1326

Phone: 508-264-5491

E-Mail: Linn@ultra.enet.dec.com

Appendix C

RFC1115 Privacy Enhancement for Internet Electronic Mail: Part III – Algorithms, Modes, and Identifiers

Network Working Group

J. Linn

Request for Comments: 1115

DEC

IAB Privacy Task Force

August 1989

Privacy Enhancement for Internet Electronic Mail:

Part III -- Algorithms, Modes, and Identifiers

STATUS OF THIS MEMO

This RFC suggests a draft standard elective protocol for the Internet

community, and requests discussion and suggestions for improvement.

This RFC provides definitions, references, and citations for algorithms, usage modes, and associated identifiers used in RFC-1113 and RFC-1114 in support of privacy-enhanced electronic mail.

Distribution of this memo is unlimited.

ACKNOWLEDGMENT

This RFC is the outgrowth of a series of IAB Privacy Task Force meetings and of internal working papers distributed for those meetings. I would like to thank the following Privacy Task Force members and meeting guests for their comments and contributions at the meetings which led to the preparation of this RFC: David Balenson, Curt Barker, Jim Bidzos, Matt Bishop, Morrie Gasser, Russ Housley, Steve Kent (chairman), Dan Nessel, Mike Padlipsky, Rob Shirey, and Steve Wilbur.

Table of Contents

1. Executive Summary	2
2. Symmetric Encryption Algorithms and Modes	2
2.1. DES Modes	2
2.1.1. DES in ECB mode (DES-ECB)	2

2.1.2. DES in EDE mode (DES-EDE)	2
2.1.3. DES in CBC mode (DES-CBC)	3
3. Asymmetric Encryption Algorithms and Modes	3
3.1. RSA	3
4. Integrity Check Algorithms	3
4.1. Message Authentication Code (MAC)	4
4.2. RSA-MD2 Message Digest Algorithm	4
4.2.1. Discussion	4
4.2.2. Reference Implementation	5
NOTES	7

Linn

[Page 1]

RFC 1115

Mail Privacy: Algorithms

August 1989

1. Executive Summary

This RFC provides definitions, references, and citations for algorithms,

usage modes, and associated identifiers used in RFC-1113 and RFC-1114 in support of privacy-enhanced electronic mail in the Internet community. As some parts of this material are cited by both RFC-1113 and RFC-1114, and as it is anticipated that some of the definitions herein may be changed, added, or replaced without affecting the citing RFCs, algorithm-specific material has been placed into this separate RFC. The text is organized into three primary sections; dealing with symmetric encryption algorithms, asymmetric encryption algorithms, and integrity check algorithms.

2. Symmetric Encryption Algorithms and Modes

This section identifies alternative symmetric encryption algorithms and modes which may be used to encrypt DEKs, MICs, and message text, and assigns them character string identifiers to be incorporated in encapsulated header fields to indicate the choice of algorithm employed. (Note: all alternatives presently defined in this category correspond to different usage modes of the DEA-1 (DES) algorithm, rather than to other algorithms per se.)

2.1. DES Modes

The Block Cipher Algorithm DEA-1, defined in ANSI X3.92-1981 [3] may

be used for message text, DEKs, and MICs. The DEA-1 is equivalent to the Data Encryption Standard (DES), as defined in FIPS PUB 46 [4]. The ECB and CBC modes of operation of DEA-1 are defined in ISO IS 8372 [5].

2.1.1. DES in ECB mode (DES-ECB)

The string "DES-ECB" indicates use of the DES algorithm in Electronic Codebook (ECB) mode. This algorithm/mode combination is used for DEK and MIC encryption.

2.1.2. DES in EDE mode (DES-EDE)

The string "DES-EDE" indicates use of the DES algorithm in Encrypt-Decrypt-Encrypt (EDE) mode as defined by ANSI X9.17 [2] for key encryption and decryption with pairs of 64-bit keys. This algorithm/mode combination is used for DEK and MIC encryption.

Linn

[Page 2]

RFC 1115

Mail Privacy: Algorithms

August 1989

2.1.3. DES in CBC mode (DES-CBC)

The string "DES-CBC" indicates use of the DES algorithm in Cipher Block Chaining (CBC) mode. This algorithm/mode combination is used for message text encryption only. The CBC mode definition in IS 8372 is equivalent to that provided in FIPS PUB 81 [6] and in ANSI X3.106-1983 [7].

3. Asymmetric Encryption Algorithms and Modes

This section identifies alternative asymmetric encryption algorithms and modes which may be used to encrypt DEKs and MICs, and assigns them character string identifiers to be incorporated in encapsulated header fields to indicate the choice of algorithm employed. (Note: only one alternative is presently defined in this category.)

3.1. RSA

The string "RSA" indicates use of the RSA public-key encryption algorithm, as described in [8]. This algorithm is used for DEK and MIC encryption, in the following fashion: the product n of a individual's selected primes p and q is used as the modulus for the RSA encryption algorithm, comprising, for our purposes, the individual's public key. A recipient's public key is used in conjunction with an associated public exponent (either 3 or $1+2^{16}$) as identified in the recipient's certificate.

When a MIC must be padded for RSA encryption, the MIC will be right-justified and padded on the left with zeroes. This is also appropriate for padding of DEKs on singly-addressed messages, and for padding of DEKs on multi-addressed messages if and only if an exponent of 3 is used for no more than one recipient. On multi-addressed messages in which an exponent of 3 is used for more than one recipient, it is recommended that a separate 64-bit pseudorandom quantity be generated for each recipient, in the same manner in which IVs are generated. (Reference [9] discusses the rationale for this recommendation.) At least one copy of the pseudorandom quantity should be included in the input to RSA encryption, placed to the left of the DEK.

4. Integrity Check Algorithms

This section identifies the alternative algorithms which may be used to compute Message Integrity Check (MIC) and Certificate Integrity Check (CIC) values, and assigns the algorithms character string identifiers for use in encapsulated header fields and within certificates to indicate the choice of algorithm employed.

Linn

[Page 3]

RFC 1115

Mail Privacy: Algorithms

August 1989

MIC algorithms which utilize DEA-1 cryptography are computed using a key which is a variant of the DEK used for message text encryption. The variant is formed by modulo-2 addition of the hexadecimal quantity FOF0F0F0F0F0F0F0 to the encryption DEK.

For compatibility with this specification, a privacy-enhanced mail implementation must be able to process both MAC (Section 2.1) and RSA-MD2 (Section 2.2) MICs on incoming messages. It is a sender option

whether MAC or RSA-MD2 is employed on an outbound message addressed to only one recipient. However, use of MAC is strongly discouraged for messages sent to more than a single recipient. The reason for this recommendation is that the use of MAC on multi-addressed mail fails to prevent other intended recipients from tampering with a message in a manner which preserves the message's appearance as an authentic message from the sender. In other words, use of MAC on multi-addressed mail provides source authentication at the granularity of membership in the message's authorized address list (plus the sender) rather than at a finer (and more desirable) granularity authenticating the individual sender.

4.1. Message Authentication Code (MAC)

A message authentication code (MAC), denoted by the string "MAC", is computed using the DEA-1 algorithm in the fashion defined in FIPS PUB 113 [1]. This algorithm is used only as a MIC algorithm, not as a CIC algorithm.

As noted above, use of the MAC is not recommended for multicast messages, as it does not preserve authentication and integrity among individual recipients, i.e., it is not cryptographically strong enough for this purpose. The message's canonically encoded text is padded at

the end, per FIPS PUB 113, with zero-valued octets as needed in order to form an integral number of 8-octet encryption quanta. These padding octets are inserted implicitly and are not transmitted with a message. The result of a MAC computation is a single 64-bit value.

4.2. RSA-MD2 Message Digest Algorithm

4.2.1. Discussion

The RSA-MD2 Message Digest Algorithm, denoted by the string "RSA-MD2", is computed using an algorithm defined in this section. It has been provided by Ron Rivest of RSA Data Security, Incorporated for use in support of privacy-enhanced electronic mail, free of licensing restrictions. This algorithm should be used as a MIC algorithm whenever a message is addressed to multiple recipients. It is also the only algorithm currently defined for use as CIC. While its continued use as the standard CIC algorithm is anticipated, RSA-MD2

may be supplanted by later recommendations for MIC algorithm selections.

The RSA-MD2 message digest algorithm accepts as input a message of any length and produces as output a 16-byte quantity. The attached reference implementation serves to define the algorithm; implementors may choose to develop optimizations suited to their operating environments.

4.2.2. Reference Implementation

```

/* RSA-MD2 Message Digest algorithm in C */

/* by Ronald L. Rivest 10/1/88 */

#include <stdio.h>

/*****

/* Message digest routines: */

/* To form the message digest for a message M */

/* (1) Initialize a context buffer md using MDINIT */

/* (2) Call MDUPDATE on md and each character of M in turn */

/* (3) Call MDFINAL on md */

```

```

/* The message digest is now in md->D[0...15] */

/*****

/* An MDCTX structure is a context buffer for a message digest */
/* computation; it holds the current "state" of a message digest */
/* computation */

struct MDCTX
{
    unsigned char D[48]; /* buffer for forming digest in */
                        /* At the end, D[0...15] form the message */
                        /* digest */

    unsigned char C[16]; /* checksum register */

    unsigned char i;     /* number of bytes handled, modulo 16 */

    unsigned char L;     /* last checksum char saved */
};

/* The table S given below is a permutation of 0...255 constructed */
/* from the digits of pi. It is a 'random' nonlinear byte */
/* substitution operation. */

int S[256] = {
    41, 46, 67, 201, 162, 216, 124, 1, 61, 54, 84, 161, 236, 240, 6, 19,
    98, 167, 5, 243, 192, 199, 115, 140, 152, 147, 43, 217, 188, 76, 130, 202,
    30, 155, 87, 60, 253, 212, 224, 22, 103, 66, 111, 24, 138, 23, 229, 18,
    190, 78, 196, 214, 218, 158, 222, 73, 160, 251, 245, 142, 187, 47, 238, 122,
    169, 104, 121, 145, 21, 178, 7, 63, 148, 194, 16, 137, 11, 34, 95, 33,

```

128,127, 93,154, 90,144, 50, 39, 53, 62,204,231,191,247,151, 3,
 255, 25, 48,179, 72,165,181,209,215, 94,146, 42,172, 86,170,198,
 79,184, 56,210,150,164,125,182,118,252,107,226,156,116, 4,241,

Linn

[Page 5]

RFC 1115

Mail Privacy: Algorithms

August 1989

69,157,112, 89,100,113,135, 32,134, 91,207,101,230, 45,168, 2,
 27, 96, 37,173,174,176,185,246, 28, 70, 97,105, 52, 64,126, 15,
 85, 71,163, 35,221, 81,175, 58,195, 92,249,206,186,197,234, 38,
 44, 83, 13,110,133, 40,132, 9,211,223,205,244, 65,129, 77, 82,
 106,220, 55,200,108,193,171,250, 36,225,123, 8, 12,189,177, 74,
 120,136,149,139,227, 99,232,109,233,203,213,254, 59, 0, 29, 57,
 242,239,183, 14,102, 88,208,228,166,119,114,248,235,117, 75, 10,
 49, 68, 80,180,143,237, 31, 26,219,153,141, 51,159, 17,131, 20,

};

/*The routine MDINIT initializes the message digest context buffer md.*/

/* All fields are set to zero.

*/

void MDINIT(md)

struct MDCTX *md;

```

{ int i;

  for (i=0;i<16;i++) md->D[i] = md->C[i] = 0;

  md->i = 0;

  md->L = 0;

}

/* The routine MDUPDATE updates the message digest context buffer to */
/* account for the presence of the character c in the message whose */
/* digest is being computed. This routine will be called for each */
/* message byte in turn. */

void MDUPDATE(md,c)

  struct MDCTX *md;

  unsigned char c;

{ register unsigned char i,j,t,*p;

  /***** Put i in a local register for efficiency *****/

  i = md->i;

  /***** Add new character to buffer *****/

  md->D[16+i] = c;

  md->D[32+i] = c ^ md->D[i];

  /***** Update checksum register C and value L *****/

  md->L = (md->C[i] ^= S[0xFF & (c ^ md->L)]);

  /***** Increment md->i by one modulo 16 *****/

  i = md->i = (i + 1) & 15;

  /***** Transform D if i=0 *****/

```

```

if (i == 0)
{
    t = 0;

    for (j=0;j<18;j++)

        /*The following is a more efficient version of the loop:*/

        /* for (i=0;i<48;i++) t = md->D[i] = md->D[i] ^ S[t]; */

        p = md->D;

        for (i=0;i<8;i++)

            { t = (*p++ ^= S[t]);

              t = (*p++ ^= S[t]);

              t = (*p++ ^= S[t]);

              t = (*p++ ^= S[t]);

              t = (*p++ ^= S[t]);

```

Linn

[Page 6]

RFC 1115

Mail Privacy: Algorithms

August 1989

```

            t = (*p++ ^= S[t]);

        }

    /* End of more efficient loop implementation */

    t = t + j;

```



```

        }

    }

}

/* The routine MDFINAL terminates the message digest computation and */
/* ends with the desired message digest being in md->D[0...15].      */
void MDFINAL(md)

    struct MDCTX *md;

    { int i,padlen;

        /* pad out to multiple of 16 */

        padlen = 16 - (md->i);

        for (i=0;i<padlen;i++) MDUPDATE(md,(unsigned char)padlen);

        /* extend with checksum */

        /* Note that although md->C is modified by MDUPDATE, character */
        /* md->C[i] is modified after it has been passed to MDUPDATE, so */
        /* the net effect is the same as if md->C were not being modified.*/

        for (i=0;i<16;i++) MDUPDATE(md,md->C[i]);

    }

/*****

/* End of message digest implementation */

*****/

```

NOTES:

- [1] Federal Information Processing Standards Publication 113,
Computer Data Authentication, May 1985.

- [2] ANSI X9.17-1985, American National Standard, Financial
Institution Key Management (Wholesale), American Bankers
Association, April 4, 1985, Section 7.2.

- [3] American National Standard Data Encryption Algorithm (ANSI
X3.92-1981), American National Standards Institute, Approved 30
December 1980.

- [4] Federal Information Processing Standards Publication 46, Data
Encryption Standard, 15 January 1977.

- [5] Information Processing Systems: Data Encipherment: Modes of
Operation of a 64-bit Block Cipher.

- [6] Federal Information Processing Standards Publication 81,
DES Modes of Operation, 2 December 1980.

Linn

[Page 7]

RFC 1115

Mail Privacy: Algorithms

August 1989

- [7] American National Standard for Information Systems - Data Encryption Algorithm - Modes of Operation (ANSI X3.106-1983), American National Standards Institute - Approved 16 May 1983.
- [8] CCITT, Recommendation X.509, "The Directory: Authentication Framework", Annex C.
- [9] Moore, J., "Protocol Failures in Cryptosystems", Proceedings of the IEEE, Vol. 76, No. 5, Pg. 597, May 1988.

Author's Address

John Linn
Secure Systems
Digital Equipment Corporation
85 Swanson Road, BXB1-2/D04
Boxborough, MA 01719-1326

Phone: 508-264-5491

EMail: Linn@ultra.enet.dec.com

Linn

[Page 8]

Appendix D

Source Code and Description – Makekeys.c

`/* Filename: makekeys.c`

`Author: Gordon D. Wishon`

`Date: 30 Mar 1990`

`Description: This file allows a user to create and store public/private
key pairs. The keys must be restored from the files using the BSAFE
module restorekey() before they can be used subsequently.`

`This file contains proprietary information of RSA Data Security, Inc., and
is used under a license granted to the U.S. Government.`

`*/`

`#include "global.h"`

```
#include "bsafebit.h"
```

```
#include <stdio.h>
```

```
#if MICROSOFTC
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <sys\types.h>
```

```
#include <sys\timeb.h>
```

```
#endif
```

```
#if THINKC
```

```
#include <unix.h>
```

```
#include <storage.h>
```

```
#endif
```

```
#if MPWC
```

```
#include <fcntl.h>
```

```
#endif
```

```
#if UNIX
```

```
#include <ctype.h>
```

```
#include <sys/time.h>
```

```
#endif
```

```
#include <math.h>

#include "bsafe.h"

#include "myclib.h"

/*****/

/* GLOBAL VARIABLES */

/*****/

BSAFE_KEY DS DigestKey={0};

BSAFE_KEY DS SecretKey={0};

BSAFE_KEY DS PublicKey={0};

BSAFE_KEY DS PrivateKey={0};


#if PROTOTYPES    /* If we should use function prototypes.*/

void savekey(char *, BSAFE_KEY BSAFE_PTR);

STATUS makedigest(BSAFE_KEY BSAFE_PTR);

STATUS makersakeys(void);

int main(void);

#else /* no PROTOTYPES */
```

```
void savekey();

STATUS makedigest();

STATUS makersakeys();

int main();


#endif /* PROTOTYPES */


void savekey(keyname,key)

char *keyname;

BSAFE_KEY BSAFE_PTR key;

{

    char ofilename[80];

    FILE *ofile;


    getoname:

    printf("Enter file name for saving %s: ", keyname);

        gets(ofilename);

        if ((ofile = fopen (ofilename,"w")) == NULL)

        {

            puts("Bad. Try again.");

            goto getoname;

        }

        fwrite((char *)key->data,1,key->size,ofile);
```



```

        fclose(ofile);
    }

STATUS makedigest(ikey)
BSAFE_KEY BSAFE_PTR ikey;
{
    int opcode;

    BSAFE_CTX ctx;

    STATUS stat;

    BSAFE_KEY BSAFE_PTR key;

    int istat, len;

    int writeval = 0;

    ULONG size;

    BYTE ibuffer[4096];

    BYTE *obuffer = NULL;

    char ofilename[80];

    FILE *ofile;

    /* Initialize context and keys */

    BSAFE_InitCtx((BSAFE_CTX BSAFE_PTR)&ctx);

    BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&DigestKey);

    DigestKey.class = BSAFE_class_DIGEST;

```

```

DigestKey.alg    = BSAFE_alg_DIG1;

DigestKey.level = 0;

getoname:

printf("\nEnter file name for saving message digest:");

    gets(ofilename);

    if ((ofile = fopen (ofilename,"w")) == NULL)

    {

        puts("Bad. Try again.");

        goto getoname;

    }


    /* Select encryption key/keys */

key = &DigestKey;

opcode = BSAFE_opcode_NULL;


    /* Now prepare to operate on the input file */


    /* Allocate buffer */

if (stat = BSAFE_ComputeSize(key,opcode,(ULONG)4096,&size))

    goto transform_exit;


obuffer = (BYTE *) malloc((UWORD) size);

```

```

    if (obuffer==NULL)
    {
        printf("\nError: buffer of size %d could not be found.",size);
        goto transform_exit;
    }

    /* Encryption loop for file data */

    len = ikey->size;

    printf ("Size of key data is %d\n", len);

    do
    {
        if (len > 4096) istat = 4096;

        else istat = len;

        memcpy (ibuffer, ikey->data, istat);

        /* Encrypt this data block (or do final call if istat==0) */

        while ((stat = BSAFE_TransformData(
            (BSAFE_CTX BSAFE_PTR)&ctx,
            key,opcode,
            (ULONG) istat, ibuffer,
            &size, obuffer))

            == ERR_BSAFE_PAUSE) printf(".") ;

        if (stat != 0) goto transform_exit;
    }

```

```
#if DEBUG

printf("\nWriting %ld bytes out on file.",size);

#endif

        writeval = fwrite((char *)obuffer,1,(int)size,ofile);

len = len - istat;

    }

    while (istat > 0);

transform_exit:

    if (obuffer!=NULL)

        free(obuffer);

    BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

fclose (ofile);

#if UNIX

    if (stat)

        unlink (ofilename);

#else

    if (stat)

        remove (ofilename);

#endif

    return(stat);

}
```

```

STATUS makersakeys()
{
    STATUS stat;

    static int keylength;

    BSAFE_CTX ctx;

    char buffer[80];

    if (MINMODBITS == MAXMODBITS)
    {
        printf ("RSA modulus length set at %d bits.\n",MINMODBITS);

        keylength = MINMODBITS;
    }

    else
    {
getklen:
        printf("Enter desired RSA key length in bits (%d--%d): ",
            MINMODBITS,MAXMODBITS);

        gets(buffer);

        sscanf(buffer," %d",&keylength); /* kill CR */

        if (keylength > MAXMODBITS || keylength < MINMODBITS)
        {

            puts ("Illegal key size.");

            goto getklen;
        }
    }
}

```

```

    }

}

printf ("\nStand by please...\n");

    initrandom();

    printf("Generating RSA key pair...");

    PublicKey.class = BSAFE_class_PUBLIC;

    PublicKey.alg    = BSAFE_alg_RSA;

    PublicKey.level = keylength;

    PublicKey.memstate = BSAFE_memstate_NULL;

    PrivateKey.memstate = BSAFE_memstate_NULL;

    BSAFE_InitCtx((BSAFE_CTX BSAFE_PTR)&ctx);

    /* Release storage in case these were previously created */

    stat = BSAFE_KeyHandler((BSAFE_KEY BSAFE_PTR)&PublicKey,

        BSAFE_opcode_FREE);

    if (stat != 0)

        return(stat);

    stat = BSAFE_KeyHandler((BSAFE_KEY BSAFE_PTR)&PrivateKey,

        BSAFE_opcode_FREE);

    if (stat != 0)

        return(stat);

    while ((stat = BSAFE_MakeKeyPair((BSAFE_CTX BSAFE_PTR)&ctx,

        (BSAFE_KEY BSAFE_PTR)&PublicKey,

        (BSAFE_KEY BSAFE_PTR)&PrivateKey))

```

```

        == ERR_BSAFE_PAUSE)

    printf(".");

    if (stat != 0)

        return(stat);

    if (stat = BSAFE_KeyHandler(

        (BSAFE_KEY BSAFE_PTR) &PublicKey,

        BSAFE_opcode_ENTRY))

        return(stat);

    if (stat = BSAFE_KeyHandler(

        (BSAFE_KEY BSAFE_PTR) &PrivateKey,

        BSAFE_opcode_ENTRY))

        return(stat);

    printf("OK.\n");

    savekey("public key",&PublicKey);

    savekey("private key",&PrivateKey);


printf ("\nIf you are using the public key in an email certificate\n");
printf ("application, you will need to create a message digest of\n");
printf ("the key, which should be placed in a separate file.\n");
printf ("\nCreate message digest?\n");


    gets(buffer);

    if (toupper (buffer[0]) == 'Y')

```

```

stat = makedigest(&PublicKey);

    return(stat);
}

```

```

int main()
{
    static int operation = 2;    /* Make rsa keys */

    STATUS stat;

    extern int BSAFE_MaxStackUsed, BSAFE_MaxStackNeeded;

    char buffer[80];

    setbuf(stdout, NULL);    /* Tell stdout to be unbuffered.*/

    BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PublicKey);

    BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PrivateKey);

    printf(
        "\nThis program will create a public/private key pair for use in");
    printf(
        "\nprotecting data.  The keys created by this program will be");
    printf(
        "\nstored in the files of your choice.  The owner should place");
    printf(

```



```
"\nthe private key in a file to which he alone has access.  If");

printf(

"\nthe private key is compromised, any data encrypted by the");

printf(

"\npublic key is in danger of compromise.\n");


BSAFE_MaxStackUsed = BSAFE_MaxStackNeeded = 0;

srand(123);

    stat = 0;

stat = makersakeys();

    printf("\nMax Stack Used = %d bytes.",BSAFE_MaxStackUsed);

    if (BSAFE_MaxStackUsed!=BSAFE_MaxStackNeeded)

        printf("\nMax Stack Needed = %d bytes.",BSAFE_MaxStackNeeded);

    printf("\nFinal status code returned = %d ",stat);

    printstatus(stat);


return (0);

} /* end of main */
```

Appendix E

Source Code and Description – Cert.c

```
/*
```

```
Filename: CERT.C
```

```
Author: Gordon D. Wishon
```

```
Date: 9 April 1990
```

```
Description: This file contains operations necessary to create new  
certificates for use in the PE-Mail system. This program contains  
material copyrighted by RSA Data Security, Inc. and is used under a  
license granted to the U.S. Government.
```

```
*/
```

```
#include <stdio.h>
```

```
#include <math.h>
```

```
#include "global.h"
```

```
#include "bsafe.h"
```

```
#include "cert.h"

#include "myclib.h"


#if MICROSOFTC

#include <dos.h>

#include <stdlib.h>

#include <string.h>

#include <sys\types.h>

#include <sys\timeb.h>

#endif


#if UNIX

#include <ctype.h>

#include <sys/time.h>

#endif


/*****/

/* GLOBAL VARIABLES */

/*****/

BSAFE_KEY DS DigestKey={0};

BSAFE_KEY DS PublicKey={0};

BSAFE_KEY DS PrivateKey={0};

cert_struct new_cert;
```

```
#if PROTOTYPES    /* If we should use function prototypes.*/

void printstatus(STATUS);

void initrandom(void);

void initcert(cert_struct *);

STATUS makedigest(CERTIFICATE *, BYTE *);

STATUS sign(BYTE *, BYTE *, ULONG *);

STATUS restorekey(void);

int main(void);

#else /* no PROTOTYPES */

void printstatus();

void initrandom();

void initcert();

STATUS makedigest();

STATUS sign();

STATUS restorekey();

int main();

#endif /* PROTOTYPES */
```

```
void printstatus(stat)

STATUS stat;

{

    switch(stat) {

        case 0: printf("OK."); break;

        case 1: printf("FALSE."); break;

        case 2: printf("ALLOCATE error."); break;

        case 3: printf("FREE error."); break;

        case 4: printf("ENTRY error."); break;

        case 5: printf("EXIT error."); break;

        case 6: printf("PAUSE."); break;

        case 7: printf("BAD KEY error."); break;

        case 8: printf("BAD CTX error."); break;

        case 9: printf("BAD OPCODE error."); break;

        case 10: printf("BAD CHECKSUM error."); break;

        case 11: printf("BAD DATA error."); break;

        case 12: printf("NEED RANDOM BYTES error."); break;

        case 13: printf("INTERNAL error."); break;

        case 14: printf("HARDWARE error or malfunction."); break;

        default: printf("UNKNOWN error code %d.",stat); break;

    }

}
```

```
void initrandom()
{
    long int i;

    #if UNIX

    struct timeval tv;

    struct timezone tz;

    #endif

    #if MICROSOFTC

    int j, k;

    struct timeb *tv;

    #endif

    /* Initialize random number generator */

    BSAFE_ResetRandom();

    printf ("\nStandby, please...\n");

    for (i=0;i<100;i++)
    {
        #if UNIX

        gettimeofday(&tv,&tz);

        BSAFE_MixInByte((BYTE)tv.tv_usec);

        #endif
    }
}
```

```
#if MICROSOFTC

ftime(tv);

BSAFE_MixInByte((BYTE)tv->millitm);

/* pause for random amount of time -- otherwise, a PC
   may return the same time in subsequent calls to ftime */

j = rand();

for (j=0; j<k; j++);

#endif

    }

}

void initcert (cert)

cert_struct *cert;

{

cert->contents.version = 0;

cert->contents.serialnum = 0;

*(cert->contents.issuer_name.country) = NULL;

*cert->contents.issuer_name.organization = NULL;

*cert->contents.issuer_name.org_unit1 = NULL;

*cert->contents.issuer_name.org_unit2 = NULL;
```

```
*cert->contents.issuer_name.org_unit3 = NULL;

*cert->contents.issuer_name.org_unit4 = NULL;

cert->contents.valid_period.start_date.day = 0;

cert->contents.valid_period.start_date.month = 0;

cert->contents.valid_period.start_date.year = 0;

cert->contents.valid_period.end_date.day = 0;

cert->contents.valid_period.end_date.month = 0;

cert->contents.valid_period.end_date.year = 0;

*cert->contents.personal_name.country = NULL;

*cert->contents.personal_name.organization = NULL;

*cert->contents.personal_name.org_unit1 = NULL;

*cert->contents.personal_name.org_unit2 = NULL;

*cert->contents.personal_name.org_unit3 = NULL;

*cert->contents.personal_name.org_unit4 = NULL;

cert->contents.pub_component.size = 0;

*cert->contents.pub_component.key_alg = NULL;

*cert->signature.signature = NULL;

cert->signature.tsize = 0;

*cert->signature.hash_alg = NULL;

}
```

```
STATUS restorekey()
```

```
{
```



```
static char ifilename[80];

int readval;

STATUS st;

BSAFE_KEY key;

BSAFE_KEY BSAFE_PTR keyp;

FILE *ifile;


BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&key);

initrandom();


getiname:

printf ("\nEnter file name:");

gets(ifilename);

if ((ifile = fopen(ifilename,"rb")) == NULL)

{

    puts("Bad. Try again.");

    goto getiname;

}

/* get file size */

printf ("Getting key size from file\n");

fseek (ifile,0L,2);

key.size = (UWORD)ftell(ifile);

rewind (ifile);
```

```

printf ("\nCalling key handler\n");

    if (st=BSAFE_KeyHandler(&key, BSAFE_opcode_ALLOCATE))

        goto exitlabel;

printf ("\nReading key data from file\n");

    if ((readval = fread

((char *)key.data,1,key.size,ifile)) < key.size)

    {

        st = ERR_BSAFE_BADKEY;

        goto exitlabel;

    }

printf ("\nCalling BSAFE_RestoreKeyData\n");

    if (st=BSAFE_RestoreKeyData(&key))

        goto exitlabel;

    /* Now move to right place */

    if (key.class == BSAFE_class_PUBLIC)

    {

        printf("\nPUBLIC ");

        keyp = &PublicKey;

    }

else if (key.class == BSAFE_class_PRIVATE)

{

printf ("\nPRIVATE ");

keyp = &PrivateKey;

```

```

    }

    else

    {

        printf("\nImproper key -- Exiting.\n");

st = 20;

return(st);

    }

    printf("key restored from file %s.\n\n", ifilename);

    keyp->size = key.size;

    keyp->data = key.data;

    keyp->handle = key.handle;

    keyp->memstate = key.memstate;

    keyp->class = key.class;

    keyp->alg = key.alg;

    keyp->level = key.level;

exitlabel:

    fclose(ifile);

    return(st);

}

STATUS sign (buffer, obuffer, tsize)

BYTE *buffer, *obuffer;

```

```

ULONG *tsize;

{

int opcode, istat, len;

BSAFE_CTX ctx;

STATUS stat;

BSAFE_KEY BSAFE_PTR key;

ULONG size;


    /* Initialize context and keys */

    BSAFE_InitCtx((BSAFE_CTX BSAFE_PTR)&ctx);


printf ("\nSigning certificate.");

key = &PrivateKey;

opcode = 1; /* encrypt */


    /* Now prepare to operate on the input buffer */


    if (stat = BSAFE_ComputeSize(key,opcode,(ULONG)4096,&size))

        goto transform_exit;


    /* Encryption loop for certificate data */

len = 16;

do

```

```

    {
istat = len;

    /* Encrypt this data block (or do final call if istat==0) */
    while ((stat = BSAFE_TransformData(
        (BSAFE_CTX BSAFE_PTR)&ctx,
        key,opcode,
        (ULONG) istat, buffer,
        &size, obuffer))

        == ERR_BSAFE_PAUSE) printf(".") ;

        if (stat != 0) goto transform_exit;

len = len - istat;
    }

    while (istat > 0);

*tsize = size;

transform_exit:

    BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

    return(stat);
}

STATUS makedigest(icert, obuffer)

CERTIFICATE *icert;

```

```

BYTE *obuffer;

{

int opcode;

    BSAFE_CTX ctx;

    STATUS stat;

BSAFE_KEY BSAFE_PTR key;

    int istat;

int len;

    ULONG size = 16;

    BYTE ibuffer[4096];


    /* Initialize context and keys */

    BSAFE_InitCtx((BSAFE_CTX BSAFE_PTR)&ctx);

    BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&DigestKey);

    DigestKey.class = BSAFE_class_DIGEST;

    DigestKey.alg    = BSAFE_alg_DIGEST;

    DigestKey.level = 0;


    /* Select encryption key/keys */

key = &DigestKey;

opcode = BSAFE_opcode_NULL;


    /* Now prepare to operate on the input buffer */

```

```

    /* Encryption loop for certificate data */

    len = sizeof(*icert);

    printf ("Size of certificate contents is %d\n", len);

    do

    {

    if (len > 4096) istat = 4096;

    else istat = len;

    memcpy (ibuffer, icert, istat);

        /* Encrypt this data block (or do final call if istat==0) */

        while ((stat = BSAFE_TransformData(

            (BSAFE_CTX BSAFE_PTR)&ctx,

            key,opcode,

            (ULONG) istat, ibuffer,

            &size, obuffer))

            == ERR_BSAFE_PAUSE) printf(".") ;

        if (stat != 0) goto transform_exit;

    len = len - istat;

    }

    while (istat > 0);

transform_exit:

```

```
    BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

    return(stat);

}
```

```
main ()

{

char buffer[80];

#ifdef MICROSOFTC

struct dosdate_t date;

#endif

BYTE *hash = HASH_ALGORITHM;

BYTE *keya = KEY_ALGORITHM;

char ofilename[80];

FILE *ofile;

STATUS stat;

    extern int BSAFE_MaxStackUsed, BSAFE_MaxStackNeeded;

    extern int TimerRate = 1;

ULONG size;

BYTE *obuffer, digest[16];


    BSAFE_MaxStackUsed = BSAFE_MaxStackNeeded = 0;

    stat = 0;
```



```
setbuf (stdout, NULL);

    BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PublicKey);

    BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PrivateKey);

initcert(&new_cert);


new_cert.contents.version = CERT_VERSION;

printf

("This program generates certificates for use in privacy\n");

printf

("enhanced mail. To successfully generate certificates, you\n");

printf

("must be authorized for access to the PRIVATE component of the\n");

printf

("issuing organization.\n");


printf

("\nEnter serial number for this certificate:");

gets (buffer);

sscanf (buffer, "%i", &(new_cert.contents.serialnum));

printf

("\nSerial number set to %d\n", new_cert.contents.serialnum);


printf
```

```
("\\nEnter organizational name of issuer:");  
  
gets (new_cert.contents.issuer_name.organization);  
  
printf  
  
("\\nEnter country abbreviation of issuer (e.g., US):");  
  
gets (new_cert.contents.issuer_name.country);  
  
printf  
  
("Organizational Unit -- Level 1 [optional]?");  
  
gets (new_cert.contents.issuer_name.org_unit1);  
  
printf  
  
("Organizational Unit -- Level 2 [optional]?");  
  
gets (new_cert.contents.issuer_name.org_unit2);  
  
printf  
  
("Organizational Unit -- Level 3 [optional]?");  
  
gets (new_cert.contents.issuer_name.org_unit3);  
  
printf  
  
("Organizational Unit -- Level 4 [optional]?");  
  
gets (new_cert.contents.issuer_name.org_unit4);  
  
printf  
  
("\\nIssuer is %s\\n", new_cert.contents.issuer_name.organization);  
  
  
#if MICROSOFTC  
  
_dos_getdate (&date);
```

```
new_cert.contents.valid_period.start_date.day =  
date.day;  
  
new_cert.contents.valid_period.start_date.month =  
date.month;  
  
new_cert.contents.valid_period.start_date.year =  
date.year;
```

```
date.year = date.year + 2;  
  
new_cert.contents.valid_period.end_date.day =  
date.day;  
  
new_cert.contents.valid_period.end_date.month =  
date.month;  
  
new_cert.contents.valid_period.end_date.year =  
date.year;
```

```
#endif
```

```
printf
```

```
("\\nThis certificate will be valid for two years from today's\\n");
```

```
printf
```

```
("date: %d/%d/%d through %d/%d/%d\\n",
```

```
new_cert.contents.valid_period.start_date.day,
```

```
new_cert.contents.valid_period.start_date.month,
```

```
new_cert.contents.valid_period.start_date.year,
```

```
new_cert.contents.valid_period.end_date.day,
new_cert.contents.valid_period.end_date.month,
new_cert.contents.valid_period.end_date.year);

printf

("Enter the subject's Distinguished Name information\n");

printf

("Country Name Abbreviation (US for United States)? ");

gets (new_cert.contents.personal_name.country);

printf

("Organizational Name? (i.e, Air Force Institute of Technology)?\n");

gets (new_cert.contents.personal_name.organization);

printf

("Organizational Unit -- Level 1 [optional]?");

gets (new_cert.contents.personal_name.org_unit1);

printf

("Organizational Unit -- Level 2 [optional]?");

gets (new_cert.contents.personal_name.org_unit2);

printf

("Organizational Unit -- Level 3 [optional]?");

gets (new_cert.contents.personal_name.org_unit3);

printf

("Organizational Unit -- Level 4 [optional]?");

gets (new_cert.contents.personal_name.org_unit4);
```

```
printf

("\nName?");

gets (new_cert.contents.personal_name.subject_name);


strcpy (new_cert.contents.pub_component.key_alg, keya);

strcpy (new_cert.signature.hash_alg, hash);


/* get applicant's public component */

printf ("\nRecovering applicant's public component.");

if (stat = restorekey()) goto exitlabel;


/* place applicant's public component into certificate */

new_cert.contents.pub_component.size = PublicKey.size;


/* key_data area only large enough for 320 bit modulus key */

/* mail program must be set up to accept keys of this size */


memcpy (new_cert.contents.pub_component.key_data,

PublicKey.data, PublicKey.size);

printf

("\nKey data is:%100.100s\n", new_cert.contents.pub_component.key_data);


/* certificate complete -- compute */
```

```
/* signature on digest of certificate */

if (stat = makedigest(&(new_cert.contents), digest)){

printf ("\nDigest computation failed -- exiting.\n");

goto exitlabel;

}


/* get issuing authority's private component */

printf

("\nPreparing to sign certificate with issuing ");

printf

("authority's private key.");

if (stat=restorekey()) goto exitlabel;


/* compute signature value */

if (stat = sign

(digest, new_cert.signature.signature, &new_cert.signature.tsize))

goto exitlabel;


printf ("\nCertificate completed.\n");

printf ("Version: %d", new_cert.contents.version);

printf ("\nSerial Number: %d", new_cert.contents.serialnum);

printf ("\nSignature: %.80c", new_cert.signature.signature);

printf ("\n%s", new_cert.signature.hash_alg);
```

```
printf ("\nIssuer Name: %s",
new_cert.contents.issuer_name.organization);

printf ("\n%s\n%s\n%s\n%s\n",
new_cert.contents.issuer_name.org_unit1,
new_cert.contents.issuer_name.org_unit2,
new_cert.contents.issuer_name.org_unit3,
new_cert.contents.issuer_name.org_unit4);

printf ("\nValid Until: %d/%d/%d",
new_cert.contents.valid_period.end_date.day,
new_cert.contents.valid_period.end_date.month,
new_cert.contents.valid_period.end_date.year);

printf ("\nCertificate issued for: %s",
new_cert.contents.personal_name.subject_name);

printf ("\n%s\n%s\n%s\n%s\n%s\n",
new_cert.contents.personal_name.organization,
new_cert.contents.personal_name.org_unit1,
new_cert.contents.personal_name.org_unit2,
new_cert.contents.personal_name.org_unit3,
new_cert.contents.personal_name.org_unit4);

getname:

printf ("\nEnter file name for storing certificate:");
```

```
    gets(ofilename);

    if ((ofile = fopen (ofilename,"w")) == NULL)
    {
        puts("Bad. Try again.");
        goto getoname;
    }

    fprintf (ofile, "%.3d", new_cert.contents.version);
    fprintf (ofile, "%.3d", new_cert.contents.serialnum);
    fprintf (ofile, "%3.3s",
new_cert.contents.issuer_name.country);
    fprintf (ofile, "%64.64s",
new_cert.contents.issuer_name.organization);
    fprintf (ofile, "%32.32s%32.32s%32.32s%32.32s",
new_cert.contents.issuer_name.org_unit1,
new_cert.contents.issuer_name.org_unit2,
new_cert.contents.issuer_name.org_unit3,
new_cert.contents.issuer_name.org_unit4);
    fprintf (ofile, "%3.3d%3.3d%4.4d",
new_cert.contents.valid_period.start_date.day,
new_cert.contents.valid_period.start_date.month,
new_cert.contents.valid_period.start_date.year);
```



```

fprintf (ofile, "%3.3d%3.3d%4.4d",
new_cert.contents.valid_period.end_date.day,
new_cert.contents.valid_period.end_date.month,
new_cert.contents.valid_period.end_date.year);

fprintf

(ofile, "%3.3s%64.64s%64.64s%32.32s%32.32s%32.32s%32.32s",
new_cert.contents.personal_name.country,
new_cert.contents.personal_name.organization,
new_cert.contents.personal_name.subject_name,
new_cert.contents.personal_name.org_unit1,
new_cert.contents.personal_name.org_unit2,
new_cert.contents.personal_name.org_unit3,
new_cert.contents.personal_name.org_unit4);

fprintf

(ofile, "%3.3d", new_cert.contents.pub_component.size);

fwrite

(new_cert.contents.pub_component.key_data, sizeof(char),100,ofile);

fprintf (ofile, "%80.80s", new_cert.signature.signature);

fprintf (ofile, "%2.2d", new_cert.signature.tsize);

fprintf (ofile, "%8.8s", new_cert.signature.hash_alg);

exitlabel:

/* close any open files, printstatus */

free(new_cert.signature.signature);

```

```
free(obuffer);  
  
    fclose(ofile);  
  
printstatus(stat);  
  
return(0);  
  
}
```

Appendix F

Source Code and Description – Pemail.c

/*

Filename: pemail.c

Author: Gordon D. Wishon

Date: 9 April 1990

Description: This program accepts a previously created text file and performs privacy enhanced mail processing. The file is then suitable for handing off to a mail User Agent program. This file contains proprietary information of RSA Data Security, Inc., and is used under a license granted to the U.S. Government.

*/

#include <stdio.h>

#include "global.h"

```
#if MICROSOFTC

#include <stdlib.h>

#include <string.h>

#include <sys\types.h>

#include <sys\timeb.h>

#endif
```

```
#if THINKC

#include <unix.h>

#include <storage.h>

#endif
```

```
#if MPWC

#include <fcntl.h>

#endif
```

```
#if UNIX

#include <ctype.h>

#include <sys/time.h>

#include <strings.h>

#endif
```

```
#include <math.h>
```

```

#include "bsafe.h"

#include "bsafebit.h"

#include "cert.h"

#include "mheaders.h"

/*****/

/* GLOBAL VARIABLES */

/*****/

/* Keys */

BSAFE_KEY DS DigestKey={0};

BSAFE_KEY DS SecretKey={0};

BSAFE_KEY DS PublicKey={0};

BSAFE_KEY DS PrivateKey={0};


/* Mail header fields */

Proc_Type X_Proc_Type = {"X-Proc-Type:", '3', ',', ' ', "ENCRYPTED"};

DEK_Info X_DEK_Info = {"X-DEK-Info:", "DES-CBC", ',', ',', NULL};

Sender_ID X_Sender_ID = {"X-Sender-ID:", NULL, ':', ' ', NULL};

Certificate X_Certificate = {"X-Certificate:", 0, NULL};

MIC_Info X_MIC_Info = {"X-MIC-Info:", NULL, ',', ' ', NULL, ',', ' ', 0, NULL};

Issuer_Certificate X_Issuer_Certificate =

{"X-Issuer-Certificate:", 0, NULL};

Recipient_ID X_Recipient_ID = {"X-Recipient-ID:", NULL, ':', ' ', NULL, ':', ' ', NULL};

```

```
Key_Info  X_Key_Info = {"X-Key-Info:", "RSA", 0, NULL};
```

```
UWORD FILESIZE = 0;
```

```
STATUS stat = 0;
```

```
#if PROTOTYPES    /* If we should use function prototypes.*/
```

```
STATUS receive(FILE *);
```

```
STATUS msgdecrypt(FILE *, BYTE *, ULONG);
```

```
STATUS recovercert(cert_struct *);
```

```
STATUS read_convert(BYTE *, FILE *, int *);
```

```
STATUS send(FILE *);
```

```
STATUS msgencrypt(BYTE *, int *, BYTE *, int *);
```

```
STATUS dekdecrypt(BYTE *, ULONG);
```

```
STATUS micencrypt(BYTE *, int *, BYTE *);
```

```
STATUS dekencrypt(BYTE *, ULONG *);
```

```
void initcert(cert_struct *);
```

```
STATUS makemic(BYTE *, int *, BYTE *);
```

```
STATUS makesdeskey(void);
```

```
int main(void);
```

```
#else /* no PROTOTYPES */
```

```
STATUS receive();

STATUS msgdecrypt();

STATUS recovercert();

STATUS read_convert();

STATUS send();

STATUS msgencrypt();

STATUS micencrypt();

STATUS dekencrypt();

STATUS dekdecrypt();

void initcert();

STATUS makemic();

STATUS makesdeskey();

int main();


#endif /* PROTOTYPES */


STATUS receive (ifile)

FILE *ifile;

/* This routine performs privacy enhancement processing on */
/* a text file. It expects an input text file which conforms */
/* to format requirements of RFC 1113. It calls routines which */
/* will parse the header fields of the input text file, extracts */
/* the encrypted, encoded Data Encryption Key (DEK), decodes the */
```

```

/* DEK, restores the private component of the recipient, uses it */
/* to decrypt the DEK, decodes the encrypted/encoded text portion */
/* of the input file, and decrypts the text. The results are */
/* placed in an output text file. */
{
FILE *ofile;

char ofilename[80], tmp[3];

BYTE *inbuff, *keybuf, *outbuff, *lastbuff;

int readval = 0, ksize, keysize;

ULONG osize, lastsize;

BYTE *foundstr;


/* Receiving: */

/* Decode from printable characters */

/* Decipher text using DEK */

/* Verify MIC quantity */

/* convert canonical form to local form */

/* print (or save) decrypted text file */


BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&SecretKey);

BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PublicKey);

BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PrivateKey);

```



```

if ((inbuff = (BYTE *) malloc (80)) == NULL) {

printf ("\nUnable to allocate space for input buffer.");

stat = 17;

goto exitlabel;

}


rewind (ifile);

/*****/

/* Parse input file for DEK field.      */

/*****/

foundstr = (BYTE *) malloc (sizeof (X_Key_Info.name));

do

{

if (!(fgets (inbuff, 80, ifile))) {

fprintf (stderr, "\nUnable to read input file");

stat = 17;

goto exitlabel;

}

readval = sizeof(X_Key_Info.name);

strncpy (foundstr, inbuff, readval);

}

while ((readval = strcmp (foundstr, "X-Key-Info:")) != 0);

```

```

/***** Get encoded/encrypted DEK size. *****/

fgets (tmp, 80, ifile);

ksize = atoi(tmp);

if ((keybuf = (BYTE *) malloc (ksize)) == NULL) {

fprintf (stderr, "\nUnable to find space for keybuf");

stat = 17;

goto exitlabel;

}

    if ((readval = fread((char *)inbuff,1,ksize,ifile)) < ksize) {

fprintf (stderr, "\nUnable to read key from input file");

stat = 17;

goto exitlabel;

}

#ifdef DEBUG

printf ("\nSize of encoded secret key is %d", ksize);

#endif

/*****

/* Decode from all printable characters. */

*****/

decode (inbuff, ksize, &keybuf, &keysize);

#ifdef DEBUG

printf ("\nSize of decoded secret key is %d", keysize);

```

```

#endif

/*****

/* Decrypt DEK.    */

*****/

if (stat=dekdecrypt(keybuf, (ULONG) keysize)) goto exitlabel;

printf ("\nDEK restored.");

/*****

/* Now read in encrypted text.    */

*****/

if (!(fgets (inbuff, 80, ifile))) {
    fprintf (stderr, "\nUnable to read input file");
    stat = 17;
    goto exitlabel;
}

nsize = 0;

outbuff = (BYTE *) malloc (0);

do
{
    if ((inbuff[0] != '\n') && (inbuff[0] != '\0')) {
        realloc (outbuff, strlen(inbuff));
        strncat (outbuff, inbuff, 62);
    }
}

```

```

}

}

/**** Read until reach special character "-", or EOF. ****/
while ((fgets (inbuff, 80, ifile)) && (!(strpbrk(inbuff, "-"))));

printf ("\nEncrypted text read.");

/* calculate size of buffer */
for (osize=1; outbuff[osize-1] != '\n'; osize++);

/*****
/* Decode from all printable characters. */
/*****
decode (outbuff, osize, &lastbuff, &lastsize);

/*****
/* Decrypt decoded encrypted text. */
/*****

getoname:

printf ("\nEnter output file name:");

gets (ofilename);

if ((ofile = fopen(ofilename, "w")) == NULL) {

puts("\nBad. Try again.\n");

```

```
goto getoname;

}

rewind (ofile);

if (stat=msgdecrypt(ofile, lastbuff, lastsize)) {

printf ("\nUnable to decrypt message.");

stat = 17;

goto exitlabel;

}

exitlabel:

if (keybuf != NULL)

free (keybuf);

if (inbuff != NULL)

free (inbuff);

if (outbuff != NULL)

free (outbuff);

if (lastbuff != NULL)

free (lastbuff);

fclose(ifile);

fclose (ofile);

return(stat);

}
```

```

STATUS recovercert(cert)

cert_struct *cert;

/* This routine builds a certificate structure in memory */
/* and fills it with data read from the input file. */
{
    FILE *cfile;

    char certfilename[80];

    char tmp1[4];

    char tmp2[5];

    /******

    /* Initialize the certificate structure. */
    /******

    initcert (cert);

    /******

    /* get recipient certificate from file */
    /******

    getcname:

    printf ("\nEnter file name of certificate:");

    gets (certfilename);

    if ((cfile = fopen(certfilename, "r")) == NULL) {

    puts("\nBad. Try again.\n");

```

```
goto getcname;

}

rewind (cfile);

fscanf (cfile, "%3c", tmp1);

cert->contents.version = atoi(tmp1);

fscanf (cfile, "%3c", tmp1);

cert->contents.serialnum = atoi(tmp1);

fscanf (cfile, "%3c", cert->contents.issuer_name.country);

fscanf (cfile, "%64c", cert->contents.issuer_name.organization);

fscanf (cfile, "%32c%32c%32c%32c",

cert->contents.issuer_name.org_unit1,

cert->contents.issuer_name.org_unit2,

cert->contents.issuer_name.org_unit3,

cert->contents.issuer_name.org_unit4);

fscanf (cfile, "%3c", tmp1);

cert->contents.valid_period.start_date.day = atoi(tmp1);

fscanf (cfile, "%3c", tmp1);

cert->contents.valid_period.start_date.month = atoi(tmp1);

fscanf (cfile, "%4c", tmp2);

cert->contents.valid_period.start_date.year = atoi(tmp2);

fscanf (cfile, "%3c", tmp1);

cert->contents.valid_period.end_date.day = atoi(tmp1);

fscanf (cfile, "%3c", tmp1);
```

```
cert->contents.valid_period.end_date.month = atoi(tmp1);

fscanf (cfile, "%4c", tmp2);

cert->contents.valid_period.end_date.year = atoi(tmp2);

fscanf (cfile, "%3c%64c%64c%32c%32c%32c%32c",

cert->contents.personal_name.country,

cert->contents.personal_name.organization,

cert->contents.personal_name.subject_name,

cert->contents.personal_name.org_unit1,

cert->contents.personal_name.org_unit2,

cert->contents.personal_name.org_unit3,

cert->contents.personal_name.org_unit4);

fscanf (cfile, "%3c", tmp1);

cert->contents.pub_component.size = atoi(tmp1);

fscanf (cfile, "%100c", cert->contents.pub_component.key_data);

fscanf (cfile, "%80c", cert->signature.signature);

fscanf(cfile, "%2c", tmp1);

cert->signature.tsize = atoi(tmp1);

fscanf (cfile, "%8c", cert->signature.hash_alg);

#if DEBUG

printf ("\nCertificate contains:\n");

printf ("\nVersion number:%d", cert->contents.version);

printf ("\nSerial number:%d", cert->contents.serialnum);

printf ("\n%3.3s", cert->contents.issuer_name.country);
```



```

printf ("\n%64.64s",
cert->contents.issuer_name.organization);

printf ("\n%32.32s\n%32.32s\n%32.32s\n%32.32s",
cert->contents.issuer_name.org_unit1,
cert->contents.issuer_name.org_unit2,
cert->contents.issuer_name.org_unit3,
cert->contents.issuer_name.org_unit4);

printf ("\n%3.3d\n%3.3d\n%4.4d",
cert->contents.valid_period.start_date.day,
cert->contents.valid_period.start_date.month,
cert->contents.valid_period.start_date.year);

printf ("\n%3.3d\n%3.3d\n%4.4d",
cert->contents.valid_period.end_date.day,
cert->contents.valid_period.end_date.month,
cert->contents.valid_period.end_date.year);

printf
("\n%3.3s\n%64.64s\n%64.64s\n%32.32s\n%32.32s\n%32.32s\n%32.32s",
cert->contents.personal_name.country,
cert->contents.personal_name.organization,
cert->contents.personal_name.subject_name,
cert->contents.personal_name.org_unit1,
cert->contents.personal_name.org_unit2,
cert->contents.personal_name.org_unit3,

```

```

cert->contents.personal_name.org_unit4);

printf ("\n%3.3d", cert->contents.pub_component.size);

printf ("\nSignature follows:");

printf ("\n%80.80c", cert->signature.signature);

printf ("\nSize of signature is %d.", cert->signature.tsize);

printf ("\n%8c", cert->signature.hash_alg);

#endif

exitlabel:

fclose (cfile);

printf ("\nCertificate recovered.");

return (stat);

}

```

```

STATUS dekdecrypt (ibuffer, isize)

BYTE *ibuffer;

ULONG isize;

/* This routine first restores the private component of the */
/* recipient, and then uses it to decrypt the DEK passed in */
/* via ibuffer. */

{

FILE *ifile;

char ifilename[80];

BYTE BSAFE_PTR buffer;

```

```

BSAFE_CTX ctx;

int len, keylen;

    int readval;

    STATUS stat;

/*****

/* Get key data from file.  */

*****/

getiname:

    printf("\nEnter file name of your PRIVATE key: ");

    gets(ifilename);

    if ((ifile = fopen(ifilename,"r")) == NULL)

    {

        puts("Bad. Try again.");

        goto getiname;

    }

    /* get file size */

    fseek (ifile,0L,2);

    PrivateKey.size = (UWORD)ftell(ifile);

    rewind (ifile);

    if (stat=BSAFE_KeyHandler(&PrivateKey, BSAFE_opcode_ALLOCATE))

        goto transform_exit;

    if ((readval = fread

```

```

((char *)PrivateKey.data,1,PrivateKey.size,ifile)) < PrivateKey.size)
{
    stat = ERR_BSAFE_BADKEY;
    goto transform_exit;
}

printf ("\nRestoring...");

if (stat=BSAFE_RestoreKeyData(&PrivateKey))
goto transform_exit;

if (PrivateKey.class != BSAFE_class_PRIVATE) {
printf ("\nNot a PRIVATE key -- terminating.");
stat = 17;
return (stat);
}

printf("\nkey restored from file %s.\n\n",ifilename);

/*****/

/* Now decrypt the DEK. */

/*****/

if (stat = BSAFE_ComputeSize(
&PrivateKey,
BSAFE_opcode_DECRYPT_CHECKSUM,
(ULONG) 256,

```

```

&keylen)) return (stat);

/* allocate output buffer */
if ((buffer = (BYTE *) malloc ((int) keylen)) == NULL) {
    fprintf (stderr, "\nUnable to allocate output buffer.");
    stat = 17;
    goto transform_exit;
}

/* Initialize context */
BSAFE_InitCtx (&ctx);

/* Decryption loop. */
len = (int) isize;
do
{
    while ((stat = BSAFE_TransformData(
        &ctx, &PrivateKey,
        BSAFE_opcode_DECRYPT_CHECKSUM,
        (ULONG) len, ibuffer,
        &keylen, buffer))
        == ERR_BSAFE_PAUSE) printf (".");
    if (stat != 0) goto transform_exit;
}

```

```
len = len - (int) isize;
}

while (len >= 0);

/* Place in proper location */

SecretKey.data = NULL;

SecretKey.class = BSAFE_class_SECRET;

SecretKey.handle = 0;

SecretKey.memstate = BSAFE_memstate_NULL;

SecretKey.alg = BSAFE_alg_DESX;

if (stat=BSAFE_MakeKeyFromKeyValue
    (&SecretKey, buffer, (UWORD) keylen))
    goto transform_exit;

transform_exit:

fclose(ifile);

if (buffer != NULL)
    free (buffer);

#ifdef DEBUG
printf ("\nSecret Key decrypted.");
#endif
```

```

BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

return(stat);

}

```

```

STATUS micencrypt (ibuffer, outsize, obuffer)

BYTE *ibuffer, **obuffer;

ULONG *outsize;

/* This routine encrypts the message digest created by the */
/* makemic routine, creating the Message Integrity Check */
/* value. The encryption is performed using the private */
/* component of ths provides a digital signature for the */
/* message being sent. */

/* ibuffer contains the 16 byte MIC value. A pointer to the */
/* results of the encryption is returned in obuffer, and */
/* size of the result is returned in outsize. */

{

int isize;

FILE *ifile;

char *ifilename;

BSAFE_CTX ctx;

BYTE keyval[16];

```

```

BYTE *buffer;

UWORD keylen;

    int readval;

    STATUS stat;

    BSAFE_KEY key;

    BSAFE_KEY BSAFE_PTR keyp;


/*****

/* Recover sender's private key. */

*****/

getiname:

    printf("\nEnter file name of sender's PRIVATE key: ");

    gets(ifilename);

    if ((ifile = fopen(ifilename,"r")) == NULL)

    {

        puts("Bad. Try again.");

        goto getiname;

    }

    /* get file size */

    fseek (ifile,0L,2);

    key.size = (UWORD)ftell(ifile);

    rewind (ifile);

    if (stat=BSAFE_KeyHandler(&key, BSAFE_opcode_ALLOCATE))

```



```

        goto transform_exit;

    if ((readval = fread
((char *)key.data,1,key.size,ifile)) < key.size)
    {

        stat = ERR_BSAFE_BADKEY;

        goto transform_exit;
    }

    if (stat=BSAFE_RestoreKeyData(&key))

        goto transform_exit;

    /* Now move to right place */

    if (key.class == BSAFE_class_PRIVATE)

        keyp = &key;

else {

    printf ("\nNot a PRIVATE key -- terminating.");

    stat = 17;

    return (stat);

}

    printf("key restored from file %s.\n\n",ifilename);

    keyp->size = key.size;

    keyp->data = key.data;

    keyp->handle = key.handle;

    keyp->memstate = key.memstate;

    keyp->class = key.class;

```

```

    keyp->alg = key.alg;

    keyp->level = key.level;

    if (stat = BSAFE_GetKeyValueFromKey
        (keyp, keyval, 16, &keylen)){
        fprintf (stderr,
            "\nError obtaining key value from PRIVATE Key.");
        goto transform_exit;
    }

    /*****

    /* Now encrypt the message digest found in ibuffer. */

    *****/

    /* allocate buffer */

    if (stat = BSAFE_ComputeSize(
        &key,
        BSAFE_opcode_DECRYPT_CHECKSUM,
        (ULONG) 16,
        outsize)) return (stat);

    /* allocate output buffer */

    if ((buffer = (BYTE *) malloc ((int) outsize)) == NULL) {
        fprintf (stderr, "\nUnable to allocate output buffer.");
    }

```

```
stat = 17;

goto transform_exit;

}

/* Initialize context */
BSAFE_InitCtx (&ctx);

isize = 16;

do
{
while ((stat = BSAFE_TransformData(
&ctx, keyp,
BSAFE_opcode_ENCRYPT_CHECKSUM,
(ULONG) isize, ibuffer,
&outsize, buffer))
== ERR_BSAFE_PAUSE) printf (".");
if (stat != 0) goto transform_exit;

isize = 0;
}

while (isize >= 0);

transform_exit:
```

```

*obuffer = buffer;

if (buffer != NULL)

free (buffer);

fclose(ifile);

BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

return(stat);

}

```

```

STATUS dekencrypt (obuffer, outsize)

BYTE *obuffer;

ULONG *outsize;

/* This routine encrypts the DEK using the public */
/* component of the recipient. The results of the */
/* encryption and its size are returned in obuffer */
/* and outsize. NOTE: The DEK has been restored */
/* from a file in a previous step. */

{

int isize;

ULONG tsize = 0;

BSAFE_CTX ctx;

BYTE keyval[16];

UWORD keylen;

```

```

printf ("\nEncrypting the SECRET Key...");

BSAFE_InitCtx (&ctx);

/*****

/* Extract the key data from the key structure. */

*****/

if (stat = BSAFE_GetKeyValueFromKey
    (&SecretKey, keyval, 16, &keylen)) {

    fprintf (stderr,
        "\nError obtaining key value from Secret Key.");
    goto transform_exit;
}

/* Encryption loop. */

isize = (int) keylen;

do
{
    while ((stat = BSAFE_TransformData(
        &ctx, &PublicKey,
        BSAFE_opcode_ENCRYPT_CHECKSUM,
        (ULONG) isize, keyval,
        outsize, obuffer))
        == ERR_BSAFE_PAUSE) printf (".");

```

```

if (stat != 0) goto transform_exit;

tsize = tsize + *outsize;

isize = isize - (int) keylen;

}

while (isize >= 0);

transform_exit:

#ifdef DEBUG

printf ("\nDone encrypting the DEK.");

#endif

*outsize = tsize;

BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

return(stat);

}

STATUS msgencrypt (ibuffer, buffsize, buffer, outsize)

BYTE *ibuffer, **buffer;

ULONG *buffsize, *outsize;

/* This routine performs the encryption of the text */
/* message. The text message and size are found at */
/* ibuffer and buffsize. The results and result size */
/* are returned in obuffer and outsize. */

{

```

```
BSAFE_CTX ctx;

ULONG isize, istat;

ULONG tsize = 0;

BYTE *obuffer;

ULONG offset = 0;


printf ("\nEncrypting ...");

if (stat = BSAFE_ComputeSize(

&SecretKey,

BSAFE_opcode_ENCRYPT_CHECKSUM,

(ULONG) 256,

outsize)) return (stat);


/* allocate temporary buffer */

if ((obuffer = (BYTE *) malloc ((int) *outsize)) == NULL) {

fprintf (stderr, "\nUnable to allocate output buffer.");

stat = 17;

return (stat);

}


/* allocate output buffer */

if ((*buffer = (BYTE *) malloc (0)) == NULL) {

fprintf (stderr, "\nUnable to allocate output buffer.");
```

```

stat = 17;

return (stat);

}

/* Initialize context */
BSAFE_InitCtx (&ctx);

/* encrypt ibuffer to buffer */
/* Encryption loop */
isize = *buffsize;

do
{
    if (isize > 256) istat = 256;

    else istat = isize;

    /* Encrypt this data block (or do final call if istat==0) */
    while ((stat = BSAFE_TransformData(
        (BSAFE_CTX BSAFE_PTR)&ctx,
        &SecretKey, BSAFE_opcode_ENCRYPT_CHECKSUM,
        istat, &ibuffer[offset],
        outsize, obuffer))

        == ERR_BSAFE_PAUSE) printf(".") ;

    if (stat != 0) goto transform_exit;

```



```

    #if DEBUG

    printf ("\noutsize is now %d", *outsize);

    #endif

    realloc (*buffer, *outsize);

    memcpy (*buffer+tsize, obuffer, *outsize);

    tsize = tsize + *outsize;

    isize = isize - istat;

    offset = offset + istat;

    }

    while (istat > 0);

transform_exit:

if (obuffer != NULL)

free (obuffer);

*outsize = tsize;

BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

return(stat);

}

STATUS read_convert(obuffer, ifile, buffsize)

BYTE **obuffer;

FILE *ifile;

ULONG *buffsize;

```

```

/* This routine reads the text message from the input file */
/* and converts it to a canonical form in accordance with */
/* RFC1113. The input text file is at ifile, and the */
/* results are placed in obuffer, the size in buffsize. */
{
    int i, j, currentsize, readval, charcount;

    BYTE *ibuffer, *tbuffer;

    UWORD FILESIZE = 0;

    BYTE tmp[1];

    /* read text file */

    clearerr(ifile);

    rewind (ifile);

    /* get file size */

    do
    {
        fread((BYTE *)tmp,1,1,ifile);

        FILESIZE++;
    }

    while (!feof(ifile));

    FILESIZE--;

    rewind (ifile);

    #if DEBUG

```

```
printf ("File size is %d\n", FILESIZE);

#endif

if ((ibuffer = (BYTE *) malloc ((int) FILESIZE)) == NULL) {

fprintf

(stderr, "\nCouldn't find space for input buffer");

stat = 17;

goto exitlabel;

}

if ((tbuffer = (BYTE *) malloc ((int) FILESIZE)) == NULL) {

fprin^f

(stderr, "\nCouldn't find space for temp buffer");

stat = 17;

goto exitlabel;

}

/* read file into ibuffer */

if ((readval =

fread((BYTE *)ibuffer,1,FILESIZE,ifile)) < FILESIZE) {

fprintf (stderr, "\nCould not read text file.");

stat = 17;

goto exitlabel;
```

```

}

/* convert to canonical form */

#ifdef DEBUG

printf ("Converting to canonical form.\n");

#endif

currentsize = (int) FILESIZE;

charcount = 0;

j = 0;

for (i=0; i<FILESIZE; i++) {

charcount++;

/* Valid ascii character ? */

if ((ibuffer[i] < '\000') || (ibuffer[i] > '\177')) {

fprintf

(stderr, "\nInput file contains invalid characters.");

fprintf (stderr, " Processing terminated.");

stat = 17;

goto exitlabel;

}

/* Check for premature EOF */

if ((ibuffer[i] == '\r') && (ibuffer[i+1] == '\n')

&& (ibuffer[i+2] == '.') && (ibuffer[i+3] == '\r'))

```

```

    && (ibuffer[i+4] == '\n')) {

fprintf

(stderr, "\nEOF encountered in input stream --");

fprintf (stderr, " processing terminated.");

stat = 17;

goto exitlabel;

}

tbuffer[j++] = ibuffer[i];

if (ibuffer[i] == '\n') { /* insert <CR> */

/* allocate additional byte for output buffer */

if ((tbuffer =

(BYTE *) realloc (tbuffer, currentsize+1)) == NULL) {

fprintf

(stderr, "\nCan't extend tbuffer --");

fprintf

(stderr, " processing terminated.");

stat = 17;

goto exitlabel;

}

j--;

tbuffer[j++] = '\r';

tbuffer[j++] = ibuffer[i];

```

```

currentsize++;

charcount = 0;

}

if (charcount >= 998) { /* insert CR-LF */
    if ((tbuffer =
        (BYTE *) realloc (tbuffer, currentsize+2)) == NULL) {
        fprintf
            (stderr, "\nCan't extend output buffer --");
        fprintf
            (stderr, " processing terminated.");
        stat = 17;
        goto exitlabel;
    }
    currentsize++;
    currentsize++;
    tbuffer[j++] = '\r';
    tbuffer[j++] = '\n';
    charcount = 0;
}

}

exitlabel:

*buffsize = (ULONG) currentsize;

#ifdef DEBUG

```

```
printf ("\nSize of tbuffer is %d", currentsize);
```

```
#endif
```

```
/* canonicalization step left out for test purposes */
```

```
/*
```

```
*obuffer = tbuffer;
```

```
*/
```

```
*obuffer = ibuffer;
```

```
return (stat);
```

```
}
```

```
STATUS send (ifile)
```

```
FILE *ifile;
```

```
/* This routine performs privacy enhancement processing on */
```

```
/* a text message. It calls routines which reads the text */
```

```
/* and converts it to canonical form in accordance with */
```

```
/* RFC1113. It then calls creates a Data Encryption Key */
```

```
/* DEK, which is used to encrypt the canonical text. Next, */
```

```
/* a digital signature is computed, and the DEK is */
```

```
/* using the recipient's public component, which is */
```

```
/* restored from the recipient's certificate. Finally, */
```

```

/* headers which conform to RFC1113 are prepended, and the */
/* output written to a text file. */

{

char buffer[80];

int i, readval;

ULONG bufsize = 0, certsize = 0, outsize = 0, lastsize = 0;

ULONG isize = 0, deksize = 0;

FILE *ofile;

char ofilename[80];

cert_struct *rcert, *scert;

extern int BSAFE_MaxStackUsed, BSAFE_MaxStackNeeded;

BYTE *canonbuffer, *outbuffer, micquantity[16];

BYTE *dekbuffer, *enc_mic, *lastbuf;

BYTE tmp1[3], tmp2[4];


BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&DigestKey);

DigestKey.class = BSAFE_class_DIGEST;

DigestKey.alg    = BSAFE_alg_DIG1;

DigestKey.level = 0;

BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&SecretKey);

BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PublicKey);

BSAFE_InitKey((BSAFE_KEY BSAFE_PTR)&PrivateKey);

```



```

/*****/

/* get recipient's certificate */

/*****/

/* allocate space for certificate */

if ((rcert=
(cert_struct *) malloc (sizeof(cert_struct))) == NULL) {

fprintf

(stderr, "\nCouldn't find space for certificate.");

stat = 17;

goto exitlabel;

}

    if (stat=recovercert(rcert)) goto exitlabel;

PublicKey.size = sizeof(rcert->contents.pub_component.key_data);

if (stat=BSAFE_KeyHandler(&PublicKey, BSAFE_opcode_ALLOCATE))

goto exitlabel;

PublicKey.data = rcert->contents.pub_component.key_data;

if (stat=BSAFE_RestoreKeyData(&PublicKey))

goto exitlabel;

printf ("\nRecipient's public key restored.");

```

```

/*****/

/* get sender's certificate */

/*****/

/*

    if (stat=recovercert(scrt)) goto exitlabel;

*/

/*****/

/* Validate recipient's certificate */

/*****/

/*****/

/* Read text file -- convert to canonical form */

/*****/

if (stat=read_convert (&canonbuffer, ifile, &buffsize))
goto exitlabel;

/*****/

/* compute message digest quantity */

/*****/

/*

if (stat=makemic (canonbuffer, &buffsize, micquantity))
goto exitlabel;

```

```

*/

/*****/

/* Create DEK from DEA-1 (Secret) key */

/*****/

if (stat=makesdeskey()) goto exitlabel;


/*****/

/* Encipher message using DEK */

/*****/

#if DEBUG

printf ("\nSize of buffer being encrypted is %d", buffsize);

#endif

if ((stat=msgencrypt

(canonbuffer, &buffsize, &outbuffer, &outsize)))

goto exitlabel;

#if DEBUG

printf ("\nSize of output from encryption is %d", outsize);

#endif


/*****/

/* Encipher DEK using recipient public component */

/*****/

```

```

/* determine size needed for buffer */

if (stat = BSAFE_ComputeSize (&PublicKey,
BSAFE_opcode_ENCRYPT_CHECKSUM,
(ULONG) 16, &deksize)) goto exitlabel;


/* allocate buffer for result */

if ((dekbuffer = (BYTE *) malloc ((int)deksize)) == NULL) {
fprintf
(stderr, "\nCould not allocate buffer for encrypted DEK.");
stat = 17;
goto exitlabel;
}

if (stat = dekencrypt (dekbuffer, &deksize)) {
fprintf (stderr, "\nEncryption failed.");
goto exitlabel;
}


/*****/

/* encrypt the MIC using sender's private component */

/*****/

/*

if (stat=micencrypt(micquantity, &X_MIC_Info.micsize, enc_mic))

```

```

goto exitlabel;

*/

/* now encode it place it in header field */

/*

if (stat=pencode

(enc_mic, &X_MIC_Info.micsize, &X_MIC_Info.MIC, &X_MIC_Info.MICsize)) {

fprintf (stderr, "\nUnable to encode MIC quantity.");

goto exitlabel;

}

*/

/*****/

/* encode sender's certificate */

/*****/

/*

certsize = sizeof (cert_struct);

if (stat=pencode

(scert, &certsize, &X_Certificate.cert, &X_Certificate.certsize)) {

fprintf (stderr, "\nUnable to encode certificate.");

goto exitlabel;

}

*/

/*****/

/* Prepend other appropriate header fields */

```

```

/*****/

getoname:

printf ("\nEnter output file name:");

gets (ofilename);

if ((ofile = fopen(ofilename, "w")) == NULL) {

puts("\nBad. Try again.\n");

goto getoname;

}

rewind (ofile);

fprintf (ofile, "\n"); /* start with blank line */

fprintf (ofile,

"-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----\n");

printf

("\nUse full Internet names in From: and To: fields.\n");

printf ("\n\nFrom:");

gets (X_Sender_ID.EntityID);

fprintf (ofile, "%12.12s", X_Sender_ID.name);

fprintf (ofile, "%-64.64s", X_Sender_ID.EntityID);

fwrite (X_Sender_ID.delimiter, sizeof(char),1, ofile);

fprintf (ofile, "%-64.64s", X_Sender_ID.IA);

fprintf (ofile, "\n");

printf ("\nTo:");

gets (X_Recipient_ID.EntityID);

```

```

fprintf (ofile, "%15.15s", X_Recipient_ID.name);

fprintf (ofile, "%-64.64s", X_Recipient_ID.EntityID);

fprintf (ofile, "\n\n");

/*****/

/*  encode encrypted DEK  */

/*****/

if (stat=pencode

(dekbuffer, deksize, &X_Key_Info.DEK, &X_Key_Info.deksize)) {

fprintf (stderr, "\nUnable to encode DEK.");

goto exitlabel;

}

fprintf (ofile, "%11.11s", X_Key_Info.name);

fprintf (ofile, "%4.4s\n", X_Key_Info.IK_Use);

fprintf (ofile, "%d\n", X_Key_Info.deksize);

fwrite (X_Key_Info.DEK, sizeof(char), X_Key_Info.deksize, ofile);

fprintf (ofile, "\n");

fprintf (ofile, "\n");

/*****/

/*  Encode encrypted message  */

/*****/

```

```

if (stat = pencode (outbuffer, outsize, &lastbuf, &lastsize)) {

fprintf (stderr,

"\nUnable to encode encrypted text.");

goto exitlabel;

}

/*****

/* Write to file for sending by mail User Agent */

/* Output lines to be limited to 64 char (plus */

/* CRLF) per RFC1113. */

*****/

do

{

if (lastsize > 62) isize = 62;

else isize = lastsize;

fwrite (lastbuf, sizeof(char), isize, ofile);

fprintf (ofile, "\n");

lastsize = lastsize - isize;

lastbuf = lastbuf+isize;

}

while (lastsize > 0);

fprintf (ofile,

"\n-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----\n");

```



```
exitlabel:

if (canonbuffer != NULL)

free (canonbuffer);

if (dekbuffer != NULL)

free (dekbuffer);

if (outbuffer != NULL)

free (outbuffer);

if (lastbuf != NULL)

free (lastbuf);

fclose(ifile);

fclose (ofile);

return(stat);

}


void initcert (cert)

cert_struct *cert;

/* This routine initializes elements of a certificate */

/* structure. */

{

cert->contents.version = 0;

cert->contents.serialnum = 0;

*(cert->contents.issuer_name.country) = NULL;
```

```
*cert->contents.issuer_name.organization = NULL;

*cert->contents.issuer_name.org_unit1 = NULL;

*cert->contents.issuer_name.org_unit2 = NULL;

*cert->contents.issuer_name.org_unit3 = NULL;

*cert->contents.issuer_name.org_unit4 = NULL;

cert->contents.valid_period.start_date.day = 0;

cert->contents.valid_period.start_date.month = 0;

cert->contents.valid_period.start_date.year = 0;

cert->contents.valid_period.end_date.day = 0;

cert->contents.valid_period.end_date.month = 0;

cert->contents.valid_period.end_date.year = 0;

*cert->contents.personal_name.country = NULL;

*cert->contents.personal_name.organization = NULL;

*cert->contents.personal_name.org_unit1 = NULL;

*cert->contents.personal_name.org_unit2 = NULL;

*cert->contents.personal_name.org_unit3 = NULL;

*cert->contents.personal_name.org_unit4 = NULL;

cert->contents.pub_component.size = 0;

*cert->contents.pub_component.key_alg = NULL;

*cert->signature.signature = NULL;

cert->signature.tsize = 0;

*cert->signature.hash_alg = NULL;

}
```

```

STATUS makemic (buffer, buffsize, quantity)

BYTE *buffer, *quantity;

ULONG *buffsize;

/* This routine creates a message digest from the text in */
/* buffer, passing it back in quantity for subsequent */
/* encryption, creating the Message Integrity Check (MIC) */
/* value, per RFC1113. */

{

int opcode;

    BSAFE_CTX ctx;

    BSAFE_KEY BSAFE_PTR key;

    ULONG len, istat;

        ULONG size = 16;


    BSAFE_InitCtx(&ctx);

        /* Select encryption key/keys */

    key = &DigestKey;

    opcode = BSAFE_opcode_NULL;


    /* Now prepare to operate on the input buffer */


    /* Encryption loop */

```

```

len = *buffsize;

do
    {
        if (len > 4096) istat = 4096;

        else istat = len;

        /* Encrypt this data block (or do final call if istat==0) */

        while ((stat = BSAFE_TransformData(

            (BSAFE_CTX BSAFE_PTR)&ctx,

            key,opcode,

            (ULONG) istat, buffer,

            &size, quantity))

            == ERR_BSAFE_PAUSE) printf(".") ;

        if (stat != 0) goto transform_exit;

        len = len - istat;

    }

    while (istat > 0);

transform_exit:

    BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);

    return(stat);

}

```

```

STATUS makesdeskey()

/* This routine creates a SECRET key, leaving the result in */
/* the global SecretKey. */
{
    initrandom();

    SecretKey.class = BSAFE_class_SECRET;

    SecretKey.alg = BSAFE_alg_DESX;

make1:

    stat = BSAFE_KeyHandler((BSAFE_KEY BSAFE_PTR)&SecretKey,
        BSAFE_opcode_FREE);

    if (stat != 0)

return(stat);

    printf("\n\nGenerating SECRET key...");

    stat = BSAFE_MakeKey((BSAFE_KEY BSAFE_PTR)&SecretKey);

    if (stat != 0)

return(stat);

    return(0);
}

int main()

/* This is the top level routine of the pemail program. */

/* It simply prompts for the name of the input text file */

```

```

/* and determines if the text is to be prepared for      */
/* sending or receiving. */
{
FILE *ifile;

static char ifilename[80];

char buffer[80];

setbuf (stdout, NULL);

getiname:

printf ("\nEnter name of text file:");

gets (ifilename);

if ((ifile = fopen(ifilename, "r")) == NULL) {

puts("\nBad. Try again.\n");

goto getiname;

}

/* Sending or Receiving? */

getresponse:

printf

("\nEnter 'S' to prepare for sending, 'D' to decrypt received message:");

gets (buffer);

if (toupper(buffer[0]) == 'S') stat = send(ifile);

else if (toupper(buffer[0]) == 'D') stat = receive(ifile);

else {

```

```

printf ("\nBad response.  Try again.\n");

goto getresponse;

}

```

```

printstatus(stat);

printf ("\n\n");

}

```

```

STATUS msgdecrypt(ofile, ibuff, isize)

FILE *ofile;          /* output file handle */

BYTE *ibuff;

ULONG isize;

/* This routine decrypts the encrypted text message.  The */
/* text will have been previously decoded from its printable */
/* form, and passed in through ibuff, along with its size in */
/* isize.  The name of the output file is passed through */
/* ofile. */

{
    BSAFE_CTX ctx;

    ULONG istat;

    ULONG size;

    ULONG offset = 0;

    BYTE *obuffer = NULL;

```

```
/* Initialize context */

BSAFE_InitCtx((BSAFE_CTX BSAFE_PTR)&ctx);

/* Now prepare to operate on the input file */

/* Allocate buffer */

if (stat = BSAFE_ComputeSize(&SecretKey,
BSAFE_opcode_DECRYPT_CHECKSUM,
(ULONG) 256, &size))

    goto transform_exit;

obuffer = (BYTE *) malloc((UWORD) size);

if (obuffer==NULL)
{
    printf("\nError: buffer of size %d could not be found.",size);

    goto transform_exit;
}

/* Encryption loop for file data */

printf ("\nDecrypting...");

do

{
```



```

if (isize > 256) istat = 256;

    else istat = isize;

    /* Decrypt this data block (or do final call if istat==0) */

    while ((stat = BSAFE_TransformData(

        (BSAFE_CTX BSAFE_PTR)&ctx,

        &SecretKey, BSAFE_opcode_DECRYPT_CHECKSUM,

        (ULONG) istat, &ibuff[offset],

        &size, obuffer))

        == ERR_BSAFE_PAUSE) printf(".") ;

    if (stat != 0) goto transform_exit;

#ifdef DEBUG
printf("\nWriting %ld bytes out on file.",size);
#endif

    fwrite((char *)obuffer,1,(int)size,ofile);

    isize = isize - istat;

    offset = offset + istat;

}

while (istat > 0);

transform_exit:

    if (obuffer!=NULL)

        free(obuffer);

```

```
BSAFE_CtxHandler((BSAFE_CTX BSAFE_PTR)&ctx,BSAFE_opcode_FREE);  
  
return(stat);  
  
}
```

Appendix G

Source Code and Description – Pesupport.c

```
#include <stdio.h>

#include "global.h"

#include <math.h>


#if MICROSOFTC

#include <stdlib.h>

#include <string.h>

#include <sys\types.h>

#include <sys\timeb.h>

#endif


#if UNIX

#include <ctype.h>

#include <sys/time.h>

#endif
```

```
#include "bsafe.h"

void printstatus(stat)

STATUS stat;

{

    switch(stat) {

        case 0: printf("OK."); break;

        case 1: printf("FALSE."); break;

        case 2: printf("ALLOCATE error."); break;

        case 3: printf("FREE error."); break;

        case 4: printf("ENTRY error."); break;

        case 5: printf("EXIT error."); break;

        case 6: printf("PAUSE."); break;

        case 7: printf("BAD KEY error."); break;

        case 8: printf("BAD CTX error."); break;

        case 9: printf("BAD OPCODE error."); break;

        case 10: printf("BAD CHECKSUM error."); break;

        case 11: printf("BAD DATA error."); break;

        case 12: printf("NEED RANDOM BYTES error."); break;

        case 13: printf("INTERNAL error."); break;

        case 14: printf("HARDWARE error or malfunction."); break;

        default: printf("UNKNOWN error code %d.",stat); break;

    }
```

```
    }  
}  
  
void initrandom()  
{  
    long int i;  
  
#if UNIX  
    struct timeval tv;  
    struct timezone tz;  
#endif  
  
#if MICROSOFTC  
    int j, k;  
    struct timeb *tv;  
#endif  
  
    /* Initialize random number generator */  
    BSAFE_ResetRandom();  
    for (i=0;i<100;i++)  
    {  
#if UNIX  
        gettimeofday(&tv,&tz);
```

```
BSAFE_MixInByte((BYTE)tv.tv_usec);
```

```
#endif
```

```
#if MICROSOFTC
```

```
ftime(tv);
```

```
BSAFE_MixInByte((BYTE)tv->millitm);
```

```
/* pause for random amount of time -- otherwise, a PC
```

```
may return the same time in subsequent calls to ftime */
```

```
j = rand();
```

```
for (j=0; j<k; j++);
```

```
#endif
```

```
}
```

```
}
```

```
/* pencode and dencode routines
```

```
Author: Gordon D. Wishon
```

```
Date: 30 April 1990
```

These routines encode and decode data per the printable encoding scheme described in RFC 1113, para. 4.3.2.4. They accept an input buffer and size, and leave the encoded/decoded text in separate output

buffers.

*/

/******

/* RFC 1113 Printable Character Used for Encoding */

/* note: '=' is used as a padding character */

/******

char map[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',

'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',

'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',

'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',

'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',

'o', 'p', 'q', 'r', 's', 't', 'u', 'v',

'w', 'x', 'y', 'z', '0', '1', '2', '3',

'4', '5', '6', '7', '8', '9', '+', '/' };

STATUS pencode (ibuffer, isize, obuffer, osize)

BYTE *ibuffer, **obuffer;

ULONG isize, *osize;

```

{

BYTE *buffer;

int i = 0, j = 0, div = 0, rem = 0, add = 0;

int tmpbuf[3];

STATUS stat = 0;

BYTE q1 = NULL, q2 = NULL;


printf ("\Encoding...");

div = isize / 3;

rem = isize % 3;

if (rem) add = 4;

*osize = (ULONG) 4*div+add;

if ((buffer = (BYTE *) malloc ((int) *osize)) == NULL) {

fprintf

(stderr, "\nCouldn't allocate space for encoding buffer.");

stat = 17;

goto exitlabel;

}


/* do integral bytes */

for (i=0; i<(div*3);) {


/* move three bytes from ibuffer to tmpbuf */

```



```
memcpy (&tmpbuf[0], ibuffer++, 1);

memcpy (&tmpbuf[1], ibuffer++, 1);

memcpy (&tmpbuf[2], ibuffer++, 1);

q1 = tmpbuf[0] >> 2;

q1 = q1 & '\077';

buffer[j++] = map[q1]; /* first output char */

q1 = tmpbuf[0] << 4;

q1 = q1 & '\060';

q2 = tmpbuf[1] >> 4;

q2 = q2 & '\017';

q1 = q1 | q2;

buffer[j++] = map[q1]; /* second output char */

q1 = tmpbuf[1] << 2;

q1 = q1 & '\074';

q2 = tmpbuf[2] >> 6;

q2 = q2 & '\003';

q1 = q1 | q2;

buffer[j++] = map[q1]; /* third output char */

q1 = tmpbuf[2] & '\077';

buffer[j++] = map[q1]; /* fourth output char */

i=i+3;

}
```

```
/* do remaining bytes */

if (rem == 1) {

    memcpy (&tmpbuf[0], ibuffer++, 1);

    q1 = tmpbuf[0] >> 2;

    q1 = q1 & '\077';

    buffer[j++] = map[q1];

    q1 = tmpbuf[0] << 4;

    q1 = q1 & '\060';

    buffer[j++] = map[q1];

    buffer[j++] = '=';

    buffer[j] = '=';

}

else if (rem == 2) {

    memcpy (&tmpbuf[0], ibuffer++, 1);

    memcpy (&tmpbuf[1], ibuffer++, 1);

    q1 = tmpbuf[0] >> 2;

    q1 = q1 & '\077';

    buffer[j++] = map[q1]; /* first output char */

    q1 = tmpbuf[0] << 4;

    q1 = q1 & '\060';

    q2 = tmpbuf[1] >> 4;

    q2 = q2 & '\017';

    q1 = q1 | q2;
```

```

buffer[j++] = map[q1]; /* second output char */
q1 = tmpbuf[1] << 2;
q1 = q1 & '\074';
buffer[j++] = map[q1]; /* third output char */
buffer[j] = '=';
}

```

```

exitlabel:

*obuffer = buffer;

return (stat);

}

```

```

STATUS decode (ibuffer, isize, obuffer, osize)

BYTE *ibuffer, **obuffer;

ULONG isize, *osize;

{

int j = 0, k = 0, p = 0, padchars = 0;

ULONG i = 0;

BYTE tmpbuf1[4], q1, q2;

BYTE *buffer;

int tmpbuf2[4];

STATUS stat = 0;

```

```

for (i=0; i<isize; i++) {

/* count number of padding characters */

if (ibuffer[i] == '=') {

padchars++;

}

}

*osize = (ULONG) isize/4*3-padchars;


/* allocate space for output buffer */

if ((buffer = (BYTE *) malloc ((int) *osize)) == NULL) {

fprintf

(stderr, "\nCouldn't allocate space for encoding buffer.");

stat = 17;

goto exitlabel;

}


/* decoding loop */

for (i=0; i<isize;) {

memcpy (tmpbuf1, &ibuffer[i], 4);

if (tmpbuf1[2] == '=') {

/* found two padding characters */

for (p=0;p<64;p++)

```

```

for (k=0;k<2;k++) {

if (tmpbuf1[k] == map[p]) tmpbuf2[k] = p;

}

q1 = tmpbuf2[0] << 2;

q1 = q1 & '\374';

q2 = tmpbuf2[1] >> 4;

q2 = q2 & '\003';

buffer[j] = q1 | q2;

}

else if (tmpbuf1[3] == '=') {

/* found one padding character */

for (p=0;p<64;p++)

for (k=0;k<=2;k++) {

if (tmpbuf1[k] == map[p]) {

tmpbuf2[k] = p;

}

}

q1 = tmpbuf2[0] << 2;

q1 = q1 & '\374';

q2 = tmpbuf2[1] >> 4;

q2 = q2 & '\003';

buffer[j++] = q1 | q2;

q1 = tmpbuf2[1] << 4;

```

```

q1 = q1 & '\360';
q2 = tmpbuf2[2] >> 2;
q2 = q2 & '\017';
buffer[j] = q1 | q2;
}

```

```

else { /* no padding characters */
for (p=0;p<64;p++)
for (k=0;k<4;k++) {
if (tmpbuf1[k] == map[p]) {
tmpbuf2[k] = p;
}
}
}

```

```

q1 = tmpbuf2[0] << 2;
q1 = q1 & '\374';
q2 = tmpbuf2[1] >> 4;
q2 = q2 & '\003';
buffer[j++] = q1 | q2;
q1 = tmpbuf2[1] << 4;
q1 = q1 & '\360';
q2 = tmpbuf2[2] >> 2;
q2 = q2 & '\017';
buffer[j++] = q1 | q2;

```

```
q1 = tmpbuf2[2] << 6;
q1 = q1 & '\300';
q2 = tmpbuf2[3] & 077;
buffer[j++] = q1 | q2;
}

i=i+4;
}

exitlabel:
*obuffer = buffer;
return (stat);
}
```

Appendix H

Source Code and Description – Header Files

```
/** PORTABILITY DEFINES *****/
```

```
/*
```

```
This can be used to turn debugging print statements on and off.
```

```
*/
```

```
#define DEBUG 0      /* 1 = debugging on. 0 = debugging off.*/
```

```
/*
```

```
The following defines tells which compiler is being used
```

```
to compile BSAFE. Generally speaking you will want to
```

```
choose to define one of these constants to 1 and all the
```

```
others to 0.
```

```
*/
```

```
/* Microsoft C (5.0) compiler for the IBM pc.*/
```

```
#define MICROSOFTC 0
```



```

/* Think's Lightspeed C (3.0) Macintosh compiler.*/

#define THINKC      0

/* Apple's MPW C (3.0) Macintosh compiler.*/

#define MPWC        0

/* Vax VMS C (2.4) compiler.*/

#define VAXC        0

/* Generic UNIX version.*/

#define UNIX        1


/*

The following compiler time switch flags the byte ordering within words.

If the bytes within a word are ordered from lo to hi then WORDBYTES_HILO
should be 0 (this is the case for the IBM-PC and other 80x86 families).

If the bytes within a word are ordered from hi to lo then WORDBYTES_HILO
should be 1 (this is the case for the 68000 family, HP's processors...)

*/

#define WORDBYTES_HILO 0    /* Byte ordering. 1 = hi-lo. 0 = lo-hi.*/


/*

The following define tells whether the compiler you are using supports
ANSI function prototypes.

*/

#if MICROSOFTC || THINKC || MPWC || VAXC

```

```

#define PROTOTYPES 1    /* Include prototype definitions.*/

#else

#define PROTOTYPES 0    /* Do not include prototype definitions.*/

#endif

/*

These are the standard simple data-type definitions BSAFE uses.

*/

#if MICROSOFTC || THINKC || MPWC || VAXC || UNIX

#define BYTE    unsigned char        /* Unsigned  8 bit.*/

#define SWORD   short int            /* Signed 16 bit.*/

#define UWORD   unsigned short int   /* Unsigned 16 bit.*/

#define SLONG   long                 /* Signed 32 bit.*/

#define ULONG   unsigned long        /* Unsigned 32 bit.*/

#endif

/*

These are standard defines that specify calling conventions and other

special features for the Microsoft C compiler on the IBM PC and compatibles.

*/

#if MICROSOFTC

#define BSAFE_CALL far pascal /* Standard Pascal calling convention.*/

#define BSAFE_PTR   far *     /* Standard pointer type assumption */

```

```

#define HANDLE      UWORD      /* Standard memory handle */

typedef UWORD      STATUS;     /* Status code ERR_BSAFE_... */

#define DS          near      /* Some things are faster if near */

#else

#define BSAFE_CALL          /* Standard C arguments calling convention.*/

#define BSAFE_PTR      *      /* Standard pointer type assumption */

#define HANDLE      UWORD      /* Standard memory handle */

typedef UWORD      STATUS;     /* Status code ERR_BSAFE_... */

#define DS          /* Meaningless for this compiler */

#endif

```

```

/** GLOBAL COMPILE-TIME CONSTANTS *****/

```

```

#ifndef TRUE /* Define this if not defined already.*/

```

```

#define TRUE 1

```

```

#endif

```

```

#ifndef FALSE /* Define this if not defined already.*/

```

```

#define FALSE 0

```

```

#endif

```

```

/*****

```

```

/* MAX???BYTES and MAX???WORDS are set at maximum capacity independant of */
/* MAXMODBITS which is used for evaluation and export limitations          */
/*****/

#define MAXMODBYTES 100

#define MAXPRMBYTES 50

#define MAXMODWORDS 50

#define MAXPRMWORDS 25

#define BSAFE_STACK_SIZE      5000 /* default stack size */

#define BSAFE_CHECKSUM_SIZE 5 /* default checksum size in bytes */

/* GLOBAL VARIABLES *****/

/* for error detection location */

extern BYTE DS BSAFE_ErrorProgram[];

extern UWORD DS BSAFE_ErrorNumber;

/* The variable BSAFE_ErrorCode is to help diagnosing the cause of specific
   error. The specified STATUS codes (ERR_BSAFE_whatever) are rather
   sparse. Whenever one of these are returned, the variables
   BSAFE_ErrorProgram and BSAFE_ErrorNumber are set to indicate the
   name of the file and a number which indicates the EXACT cause of the

```

error. The macro BSAFE_Error(num) sets these two variables,
assuming that the variable PROGRAM_NAME has been defined.

```

*/

#define BSAFE_Error(num) { char *p = PROGRAM_NAME, \
                          *q = (char *) BSAFE_ErrorProgram;\
                          do {*q++ = *p;} while (*p++); \
                          BSAFE_ErrorNumber = num; \
                          BSAFE_ErrorLog((BYTE BSAFE_PTR)PROGRAM_NAME,num);}

/* for RSA/bignum computations */

#define bignum unsigned short

extern bignum DS B_N[]; /* N = p*q */
extern bignum DS B_E[]; /* Encryption exponent */
extern bignum DS B_P[]; /* Prime p */
extern bignum DS B_Q[]; /* Prime q */
extern bignum DS B_D[]; /* Decryption exponent */
extern bignum DS B_DP[]; /* Decryption exponent mod p-1 */
extern bignum DS B_DQ[]; /* Decryption exponent mod q-1 */
extern bignum DS B_CR[]; /* CRT coefficient = inverse of Q modulo P */
extern int DS B_PSIZEBITS; /* Prime Size in bits */
extern int DS B_PSIZEBYTES; /* Prime Size in bytes */
extern int DS B_PSIZEWORDS; /* Prime Size in words */

```

```

/* RSA encryption formatting parameters */

#define RSA_MAC_BYTES 2 /* Number of bytes of mac per RSA block */

#define RSA_RAND_BYTES 5 /* Number of random bytes per RSA block */

#define RSA_HEADER_BYTES 1 /* Number of header bytes per RSA block */

#define RSA_HEADER_VALUE 11 /* Value to put in header byte for BSAFE */


/* For context usages */

typedef struct {

    UWORD maxsize; /* maximum size of stack allowed, in bytes */

    UWORD currentsize; /* current number of bytes used */

    BYTE data[100]; /* stack contents; actual length may vary */

} BSAFE_STACK;


extern BSAFE_STACK BSAFE_PTR DS BSAFE_stack; /* main stack */


/*****

/*      compile time constants for determining if DES or RC2 are present */

*****/

#define DES_PRESENT 1

#define SX1_PRESENT 0

```

```

/*****/

/* compile time constants for limiting the SX1 secret algorithm key size */

/*****/

#define SX1KEY_MAXBITS 64

#define SX1KEY_MINBITS 2

/*****/

/* Compile time constants for lseek() function (defined for Microsoft) */

/*****/

#define SEEK_SET 0

#define SEEK_CUR 1

#define SEEK_END 2

/*****/

/* Global variable for determining Encryption Exponent size in words */

/*****/

extern UWORD DS BSAFE_EE_SIZEWORDS;

/*****/

/* compile time constants for enabling testing and storage of D decryption */

```

```
/* exponent and testing and storage of CHINESE REMAINDER coefficients */  
  
/* D mod P-1, D mod Q-1, inv P mod Q, prime P and prime Q      */  
  
/*****  
  
#define ENABLE_DD 0  
  
#define ENABLE_CR 1  
  
/*****
```



```
/* Filename: bsafebit.h
```

```
Description: This file contains the definitions for maximum and  
minimum key sizes created using bsafe routines. */
```

```
/***/
```

```
/* #define MINMODBITS 256      Standard BSAFE minimum      */
```

```
#define MINMODBITS 320 /* Minimum for pilot PE-Mail implementation */
```

```
#define MAXMODBITS 320 /* Minimum for pilot PE-Mail implementation */
```

```
/* #define MAXMODBITS 632      Standard PE-Mail maximum      */
```

```
/* #define MAXMODBITS 720 Standard BSAFE maximum      */
```

```
/* Filename: CERT.H
```

```
Author: Gordon D. Wishon
```

```
Description: This file contains global declarartions for PE Mail
certificate generation and validation. Certificate generation and
validation make use of software copyrighted by RSA Data Security,
Inc. and is used under a license granted to the US Government.
```

```
Created: 4 April 1990
```

```
*/
```

```
#define HASH_ALGORITHM "RSA-MD2"
```

```
#define KEY_ALGORITHM "RSA"
```

```
#define CERT_VERSION 0
```

```
/******
```

```
Certificate Data Structure -- See RFC 1114 for format and contents
```

```
*****
```

```
typedef struct
```

```
{ BYTE signature[80]; /* one-way hash results */
```

```
ULONG tsize; /* size of signature field */
```

```
BYTE hash_alg[8];      /*name of algorithm
```

```
    used for one-way hash */
```

```
} cert_signature;
```

```
typedef BYTE cert_version;
```

```
typedef BYTE cert_serialnum;
```

```
typedef struct
```

```
{ BYTE country[3];
```

```
  BYTE organization[64];
```

```
  BYTE org_unit1[32];
```

```
  BYTE org_unit2[32];
```

```
  BYTE org_unit3[32];
```

```
  BYTE org_unit4[32];
```

```
} cert_issuer_name;
```

```
typedef struct
```

```
{ BYTE day;
```

```
  BYTE month;
```

```
  UWORD year;
```

```
} vdate;
```

```
typedef struct
```

```
{ vdate start_date;

vdate end_date;

} cert_valid_period;
```

```
typedef struct

{ BYTE country[3];

BYTE organization[64];

BYTE subject_name[64];

BYTE org_unit1[32];

BYTE org_unit2[32];

BYTE org_unit3[32];

BYTE org_unit4[32];

} cert_personal_name;
```

```
typedef struct

{ ULONG size; /* size of key in bytes -- non-standard field; not
needed if ASN1 encoding used */

BYTE key_data[100]; /* 100 bytes corresponds to 320 bit modulus
size -- non standard field */

BYTE key_alg[4];

} cert_pub_component;
```

```
typedef struct
```

```
{ cert_version version;  
  
cert_serialnum serialnum;  
  
cert_issuer_name issuer_name;  
  
cert_valid_period valid_period;  
  
cert_personal_name personal_name;  
  
cert_pub_component pub_component;  
  
} CERTIFICATE;
```

```
typedef struct  
  
{ CERTIFICATE contents;  
  
cert_signature signature;  
  
} cert_struct;
```

```
/*Filename: mheaders.h
```

```
Author:  Gordon D. Wishon
```

```
Date: 1 May 1990
```

This file contains the definition and format of the pemail header fields as described in RFC 1113.

```
*/
```

```
/******
```

```
/*Enclosing Headers*/
```

```
/******
```

```
typedef struct
```

```
{ BYTE name[13];
```

```
  BYTE subfield1;
```

```
  BYTE delimiter;
```

```
  BYTE subfield2[10];
```

```
} Proc_Type;
```

```
typedef struct
```

```
{ BYTE name[11];
```

```
  BYTE alg[8];
```

```
BYTE delimiter;
```

```
BYTE vector[16];
```

```
} DEK_Info;
```

```
typedef struct
```

```
{ BYTE name[12];
```

```
BYTE EntityID[64];
```

```
BYTE delimiter;
```

```
BYTE IA[64];
```

```
} Sender_ID;
```

```
typedef struct
```

```
{ BYTE name[14];
```

```
ULONG certsize; /* non standard field */
```

```
BYTE *cert;
```

```
} Certificate;
```

```
typedef struct
```

```
{ BYTE name[11];
```

```
BYTE alg[7];
```

```
BYTE delimiter1;
```

```
BYTE encralg[3];
```

```
BYTE delimiter2;
```

```
ULONG MICsize; /* non standard field */
```

```
BYTE *MIC;
```

```
} MIC_Info;
```

```
typedef struct
```

```
{ BYTE name[21];
```

```
ULONG certsize; /* non standard field */
```

```
BYTE *cert;
```

```
} Issuer_Certificate;
```

```
typedef struct
```

```
{ BYTE name[15];
```

```
BYTE EntityID[64];
```

```
BYTE delimiter1;
```

```
BYTE IA[64];
```

```
BYTE delimiter2;
```

```
BYTE version;
```

```
} Recipient_ID;
```

```
typedef struct
```

```
{ BYTE name[11];
```

```
BYTE IK_Use[4];
```

```
BYTE delimiter1;
```



```
ULONG deksize; /* non standard field */  
  
BYTE delimiter2;  
  
BYTE *DEK;  
  
} Key_Info;
```

Bibliography

- [1] Abrams, Marshall D. and Podell, Harold J., *Computer and Network Security*, IEEE Computer Society Press, Washington, D.C., 1987.

- [2] Summers, R.C., *An Overview of Computer Security*, IBM Systems Journal, Vol 23, No 4, 1984, International Business Machines Corp., 1984.

- [3] Bell, David E. and La Padula, Leonard J., *Secure Computer Systems: Unified Exposition and Multics Interpretation, Foundations, and Model*, The MITRE Corp., Bedford, Mass. MTR-2997 Rev 1, March 1976.

- [4] Rushby, John, *Networks are Systems*, from Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, 1985.

- [5] National Computer Security Center, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, National Computer Security Center, NCSC-TG-005, 31 July 1987.

- [6] Voydock, Victor L. and Kent, Stephen T., *Security in High-Level Network Protocols*, IEEE Communications Magazine, July 1985, The Institute of Electrical and Electronic Engineers, Inc., 1985.

- [7] Merkle, Ralph C., *Secrecy, Authentication, and Public Key Systems*, UMI Research Press, Ann Arbor, Michigan, 1982.

- [8] Analysis Communications Systems Corp., *Everything You Wanted to Know About DES Security But Did Not Know Who to Ask*, 1984, Analysis Communications Systems Corp., 1984.

- [9] Diffie, W., and Hellman, M., *New Directions in Cryptography*, IEEE Trans. Inform. Theory IT-22, 6 (Nov 1976), 644-654.

- [10] Diffie, W., *The First Ten Years of Public-Key Cryptology*, Proceedings of the IEEE, Vol. 76, No. 5, May 1988.

- [11] Rivest, R., Shamir, A., and Adelman, L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol. 21, No. 2, Feb 1978.